

HZ BOOKS
华章教育

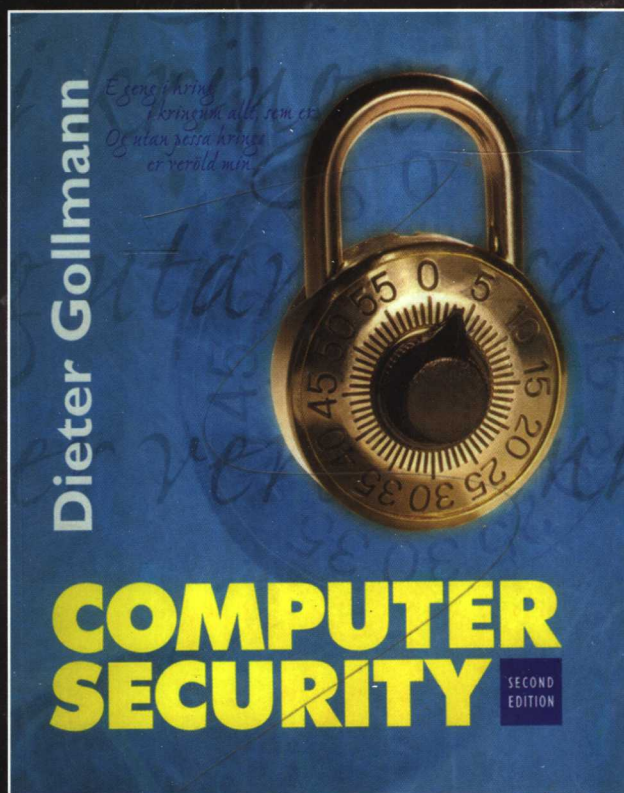


计 算 机 科 学 丛 书


原书第2版

计算机安全学

(德) Dieter Gollmann 著 张小松 等译
汉堡科技大学 成都电子科技大学



Computer Security
Second Edition

 机械工业出版社
China Machine Press

计算机安全学 (原书第2版)

本书第1版评论

“对那些陷于计算机安全痛苦问题的人而言，这是一本非常有用的书，我把它推荐给我的学生。”

—Antonia Jones, 英国加地夫大学

如今，安全不再仅仅是专家们感兴趣的话题，对此感兴趣的还包括所有的终端用户、系统管理员以及软件开发人员。

总有特殊的安全威胁会伴随着新的IT应用软件的开发而出现。面对永无休止的挑战，传统的解决办法也许不但不能解决问题，反而会恶化这些问题。因而，我们需要新的解决办法。

本书提出了与计算机安全有关的全面而简练的总的看法，帮助读者应对各个级别的安全问题。本书从基本的定义和概念开始，进而勾画出计算机系统的核心机制。本书覆盖了网络、操作系统以及数据库的安全问题，并显示了应该如何评估及解决安全问题。

本书第2版包括了有关软件安全的新章节、分布式系统的认证、访问控制中新的范例以及移动等方面的内容，同时还涵盖了对安全管理和密码学的简要介绍。

本书对计算机科学、工程学及相关学科下学习计算机或信息安全课程的本科生或研究生而言，是十分必要的读物。而对于技术和工程管理人员而言，这本书也会为解决复杂的安全难题提供很好的切入点。

本书既适合课堂教学，也适合自学。教学幻灯片、每章练习的解题方案等附加的资源都可以在 www.wiley.com/go/gollmann 上找到。

作者简介

Dieter Gollmann

汉堡科技大学分布式应用软件安全的教授，曾做过伦敦大学皇家豪乐威学院的访问教授、丹麦技术大学副教授。另外，他曾在剑桥大学微软研究中心担任研究信息安全的研究员。



www.wiley.com

投稿热线: (010) 88379604
购书热线: (010) 68995259, 68995264
读者信箱: hzjsj@hzbook.com

华章网站 <http://www.hzbook.com>

网上购书: www.china-pub.com

封面设计: 邹勇 林杉



上架指导: 计算机/计算机安全

ISBN 978-7-111-22864-6



9 787111 228646

定价: 29.00元

计 算 机

TP309/117

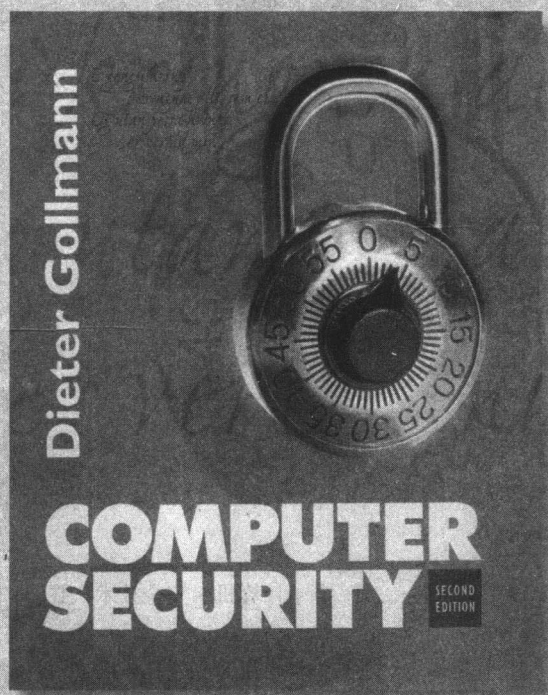
2008

书

原书第2版

计算机安全学

(德) Dieter Gollmann 著 张小松 等译
汉堡科技大学 成都电子科技大学



Computer Security
Second Edition



机械工业出版社
China Machine Press

本书源于作者的课程讲义, 现已升级至第2版。书中涉及计算机安全的许多重要的历史事件, 重点讲解了终端系统的技术安全问题, 并在新版中新增了分布式系统的认证、移动、软件安全等章节。主要内容包括: 计算机安全基础、身份识别与认证、访问控制、引用监控器、UNIX 安全、Windows 安全、Bell-LaPadula 模型、安全模型、安全评估、密码学、分布式系统中的认证、网络安全、软件安全、新的访问控制范例、移动、数据库安全等内容。

Dieter Gollmann: Computer Security, Second Edition (ISBN: 0-470-86293-9)

Authorized translation from the English language edition published by John Wiley & Sons, Inc.

Copyright © 2006 by John Wiley & Sons, Inc.

All rights reserved.

本书中文简体字版由约翰·威利父子公司授权机械工业出版社独家出版。未经出版者书面许可, 不得以任何方式复制或抄袭本书内容。

版权所有, 侵权必究。

本书法律顾问 北京市展达律师事务所

本书版权登记号: 图字: 01-2006-5288

图书在版编目(CIP)数据

计算机安全学(原书第2版)/(英)高尔曼(Gollmann, D.)著; 张小松等译. —北京: 机械工业出版社, 2008. 4

书名原文: Computer Security, Second Edition

(计算机科学丛书)

ISBN 978-7-111-22864-6

I. 计… II. ①高… ②张… III. 电子计算机—安全技术 IV. TP309

中国版本图书馆 CIP 数据核字(2007)第 178801 号

机械工业出版社(北京市西城区百万庄大街 22 号 邮政编码 100037)

责任编辑: 王 璐

北京市慧美印刷有限公司印刷·新华书店北京发行所发行

2008 年 4 月第 1 版第 1 次印刷

184mm×260mm·14.75 印张

标准书号: ISBN 978-7-111-22864-6

定价: 29.00 元

凡购本书, 如有倒页、脱页、缺页, 由本社发行部调换

本社购书热线: (010)68326294

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域中取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与 Prentice Hall, Addison-Wesley, McGraw-Hill, Morgan Kaufmann 等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出 Tanenbaum, Stroustrup, Kernighan, Jim Gray 等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及度藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近260个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”。为了保证这两套丛书的权威性，同时也为了更好地为学校和老师服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这两套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业的教学度身订造的。其中许多教材均已为 M. I. T., Stanford, U. C. Berkeley, C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，

而且各具特色—有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzjsj@hzbook.com

联系电话：(010)68995264

联系地址：北京市西城区百万庄南街1号

邮政编码：100037

专家指导委员会

(按姓氏笔画顺序)

尤晋元
石教英
张立昂
邵维忠
周克定
郑国梁
高传善
裘宗燕

王珊
吕建
李伟琴
陆丽娜
周傲英
施伯乐
梅宏
戴葵

冯博琴
孙玉芳
李师贤
陆鑫达
孟小峰
钟玉琢
程旭

史忠植
吴世忠
李建中
陈向群
岳丽华
唐世渭
程时端

史美林
吴时霖
杨冬青
周伯生
范明
袁崇义
谢希仁

译者序

计算机的安全问题已经成为世界性关注的问题,计算机用户在使用计算机的过程中几乎无一幸免地受到过各种病毒、恶意软件、流氓软件的骚扰或攻击。这些攻击轻则造成系统的工作不稳定,影响工作的正常进行;重则造成系统崩溃,保密资料被窃。据统计,电脑病毒对全世界企业造成的损失呈逐年上升趋势。2001年造成的损失为110亿美元,但到了2003年造成的损失已达到550亿美元。新病毒,新攻击的破坏方式层出不穷。无论反病毒公司的动作有多快,他们始终滞后于病毒发展的速度,可见,解决计算机安全问题已经迫在眉睫。从普通用户的角度讲,树立计算机安全使用意识是必须解决的问题;从计算机安全专业技术人员的角度讲,设计并实现更加安全的计算机系统架构也是目前安全现状提出的进一步要求。而这两者都必须从源头入手,即需要了解计算机安全的基本概念和原理,本书即是从技术原理的角度介绍了计算机安全领域最基本和最重要的原理,对于希望全面掌握了解计算机安全原理的人而言,这是一本非常有用的书。

全书分为17章,从计算机安全的基本的定义和概念开始,覆盖了网络安全、操作系统安全以及数据库安全等方面的主题,内容几乎涉及了目前计算机安全领域的所有方面,从基于口令访问控制的基本模型到操作系统完整性和内存访问控制保护机制,从计算机安全模型到计算机安全评估,从密码学和密钥管理协议到网络安全,从软件漏洞到基于代码的访问控制,从移动服务安全机制到数据库安全,内容丰富,深入浅出。

本书适合作为高等学校信息安全专业本科或工程硕士的《信息安全概论》、《计算机安全》等基础课程的参考教材,也可作为其他学科在安全类课程方面的参考教材,还可以作为信息专业技术人员的参考用书,是信息安全方面较理想的教学参考书籍。

本书的翻译工作由潘小会、邵林、王巧、谢飞、刘飞、陈亮、刘智、王东、龙小书、姜毅完成,最终由张小松统稿。

在翻译过程中,译者在不偏离作者原意的原则下,尽量运用流畅、准确的文句表达原书内容。由于译者水平有限,难免存在译误,敬请读者指正。

译者

2007年12月

前言

本书源于笔者的课程讲义，笔者曾在有关信息安全的为期一年的研究生项目中讲授。在这个项目头一年讨论产业配置的引用术语时，主管安全的管理者强调了阐明非项目目标的重要性。因此我们将非课程目标列举如下：

- 这不是一本计算机安全手册
- 这不是关于计算机安全的百科全书
- 这不是计算机安全的历史书

当然，本书会参考计算机安全中的历史事件，讲到很多背景知识。书中涵盖该领域中的众多材料，笔者希望安全实施人员会从中得到有用的观点和内容。不过，最初本书是作为计算机安全教科书而写的，主旨是为那些有计算机科学背景的学生提供一些评定和比较安全产品技术优势的基础。

本书第1版的前言也作了上述介绍，接着解释了笔者所选择的主题和书的结构。第1版的材料编于数年前，这段时间里因特网已经对商业用途开放，在此之前，万维网也早已经存在了，但这些事件带来的变化还没有广及今天的程度。回顾一下，从20世纪90年代中期至90年代末的这一段时间里，很多人都相信通信安全特别是强密码学可以保障因特网和万维网的安全访问。

如今，通过广泛部署令人满意的通信安全协议，这些问题已经得到解决。但是，安全仍然是一个难题。人们的关注又回到了终端系统。缓冲区溢出、一般性的代码漏洞引起了人们的注意。访问控制不再是主要由操作系统实施，中间层和应用层也都实施了访问控制。基于代码的访问控制已经替代了传统的基于用户身份的替代。

本书第2版试图客观评价计算机安全领域中的这些发展，并增加了分布式系统的认证、移动、软件安全以及访问控制中新的范例等章节。Unix安全、Windows 2000安全、网络安全等章节的内容都已经更新。另外，本书还重新安排了安全模型方面的材料，并将其分为两章。第8章讲述了如何构建一个形式化的安全模型，比如Bell-LaPadula模型，以及如何将它应用于计算机系统的分析。第9章说明其他重要安全模型的概况。时下流行的一些问题，比如可信计算，本书只略提一二。其他方面，如网络安全服务，在这一版中删掉了，因为(但不局限于此)这个领域仍在不断发展，技术细节很快就过时了！

与本书第1版一样，第2版也很关注计算机安全，如现在信息技术应用中使用的终端系统的技术安全问题。经过深思熟虑，笔者决定将重点放在技术方面。总的来说，这不是一本有关信息安全的书，虽然书中包括了风险分析和安全管理概论性章节。这些章节可供那些教授信息安全课程的讲师作为背景教材使用。本书没有在这些章节上长篇大论并不意味着它们对计算机安全不重要。相反，没有技术安全应用的参考背景，就不能制定安全决策；如果管理不当，或者部署错误，即使最好的技术措施也会失效。在上面提到的研究生项目中，有关技术主题的课程就是与安全管理方面的课程一起讲授的，这些有关安全管理的课程会讲到围绕计算机安全的非技术安全问题。即使在关注技术问题时，如果在本书讲完后，读者在搞清系统目的前仍把“这个系统是安全的吗”这个问题当成一个有意义的问题，那么笔者会觉得自己很失败。

在整本书中，笔者始终尝试着从书中各方面的主题讨论中提取出通用的安全原则。在讲述

具体的安全系统时，也展示了诸如 Unix 和 Windows 2000 中的安全特征。它们主要作为这些一般性观点的例证，而不作为对这些系统的全面介绍。其一，这是由于篇幅的限制，很多章节涉及的主题都可以单独写成一本书；再者是因为计算机安全系统的不断变化的特质，对技术细节的解释很快会过时；最后是因为这样一个经常会听到的抱怨：安全实施者已悉知系统是如何工作的，但是并不知道如何让它们更好地工作。因此，本书尝试提醒读者如何让系统工作得更出色。

本书的每一章都配置了练习，但笔者不能说对所有的练习都很满意。不能说计算机安全像一堆菜谱一样，可以在一本典型的教材练习限定的范围内逐一展示完全。在某些领域中，如密码安全或密码学，编写有精确答案的练习十分容易，而且只要执行正确的操作步骤即可找出准确答案。有些领域则更适合项目、论文或讨论。虽然，大家很自然地会联想到提供一个有实际系统实验的计算机安全课题，但笔者在本书中并没有提出实验环节的建议。操作系统、数据库管理系统或者防火墙是实践练习的主要来源。讲师可根据实际工作中使用的特定系统收集具体实例。针对具体的系统，还有很多优秀的讲解如何使用系统的安全机制的书籍。

由于这是一本教材，所以有时候我会在练习中收集一些很重要的材料，这些材料可能在一些计算机安全手册的主体部分有收录。

笔者很感谢那些鼓励本书出版并为第 1 版提供反馈意见的同事们，他们对本书中哪些地方很有用，哪些地方用处不大提出了合理建议。然而，并非所有的建议都被第 2 版采纳。笔者还要特别感谢 Jason Crampton，他提了很多详细而有用的建议，并修正了很多章节。感谢 Tuomas Aura 和 Kai Rannenberg，笔者曾就一些章节中的专业技术问题向他们讨教过，他们提供了很详细的建议。

Dieter Gollmann

2005 年 8 月于汉堡

目 录

出版者的话	
专家指导委员会	
译者序	
前言	
第1章 绪论	1
1.1 攻击与攻击者	1
1.2 安全	2
1.3 安全管理	3
1.3.1 安全策略	3
1.3.2 测量安全	4
1.3.3 标准	5
1.4 风险与威胁分析	5
1.4.1 资产	6
1.4.2 漏洞	6
1.4.3 威胁	7
1.4.4 风险	8
1.4.5 对策——减轻风险	8
1.5 深层阅读	9
1.6 练习	9
第2章 计算机安全基础	10
2.1 定义	10
2.1.1 安全	10
2.1.2 计算机安全	11
2.1.3 机密性	11
2.1.4 完整性	12
2.1.5 可用性	13
2.1.6 问责性	13
2.1.7 不可否认性	14
2.1.8 可靠性	14
2.1.9 计算机安全定义	14
2.2 计算机安全进退两难的困境	15
2.3 数据与信息	15
2.4 计算机安全原则	16
2.4.1 控制重点	16
2.4.2 人一机标尺	17
2.4.3 复杂性 with 保证性	18
2.4.4 集中控制 or 分布控制	18
2.5 下层	18
2.6 深层阅读	19
2.7 练习	20
第3章 身份识别与认证	21
3.1 用户名与口令	21
3.2 口令管理	22
3.3 选择口令	22
3.4 欺骗攻击	24
3.5 保护口令文件	25
3.6 一次签到	26
3.7 可供选择的方法	27
3.8 深层阅读	28
3.9 练习	29
第4章 访问控制	30
4.1 背景	30
4.2 认证和授权	30
4.3 访问操作	32
4.3.1 访问模式	32
4.3.2 Bell-LaPadula 模型的访问权限	32
4.3.3 当前的操作系统	33
4.4 所有权	34
4.5 访问控制结构	34
4.5.1 访问控制矩阵	34
4.5.2 能力	34
4.5.3 访问控制列表	35
4.6 中间控制	36
4.6.1 组和否定的许可	36
4.6.2 特权	36
4.6.3 基于角色的访问控制	37
4.6.4 保护环	37
4.7 偏序	38
4.7.1 VSTa 微内核中的能力	39
4.7.2 安全级别的格	39
4.7.3 多级安全	40
4.8 深层阅读	41
4.9 练习	41

第 5 章 引用监控器	42	6.7.1 管理超级用户	64
5.1 引言	42	6.7.2 可信主机	65
5.1.1 部署引用监控器	43	6.7.3 审计日志与入侵检测	65
5.1.2 执行监控器	43	6.7.4 安装与配置	65
5.2 操作系统完整性	43	6.8 深层阅读	66
5.2.1 操作模式	44	6.9 练习	66
5.2.2 受控调用	44	第 7 章 Windows 2000 安全	68
5.3 硬件安全特性	44	7.1 引言	68
5.3.1 安全基本原理	44	7.1.1 体系结构	68
5.3.2 计算机体系结构的简单概述	45	7.1.2 注册表	69
5.3.3 进程和线程	46	7.1.3 域	70
5.3.4 受控调用——中断	46	7.1.4 活动目录	70
5.3.5 Intel 80386/80486 上的保护	47	7.2 访问控制——组件	71
5.4 存储器保护	49	7.2.1 主角	71
5.5 深层阅读	52	7.2.2 主体	72
5.6 练习	52	7.2.3 对象	73
第 6 章 UNIX 安全	53	7.2.4 访问掩码	75
6.1 引言	53	7.2.5 扩展权限	75
6.2 主角	54	7.3 访问决策	75
6.2.1 用户账户	54	7.3.1 DACL	75
6.2.2 超级用户(根)	55	7.3.2 决策算法	76
6.2.3 组	55	7.3.3 ACE 继承	77
6.3 主体	55	7.4 受限上下文	79
6.3.1 登录和口令	56	7.5 管理	80
6.3.2 影子口令文件	56	7.5.1 用户账户	80
6.4 对象	56	7.5.2 默认用户账户	80
6.4.1 i 节点	57	7.5.3 审计	81
6.4.2 默认许可位	57	7.5.4 小结	81
6.4.3 目录的许可	58	7.6 深层阅读	82
6.5 访问控制	58	7.7 练习	82
6.5.1 设置 UID 和 GID	59	第 8 章 Bell-LaPadula 模型	83
6.5.2 更改许可	59	8.1 状态机模型	83
6.5.3 UNIX 访问控制的不足	60	8.2 Bell-LaPadula 模型	84
6.6 一般安全原则的实例	60	8.2.1 状态集	84
6.6.1 受控调用的应用	61	8.2.2 安全策略	84
6.6.2 删除文件	61	8.2.3 基本安全定理	86
6.6.3 设备保护	61	8.2.4 稳定性	86
6.6.4 改变文件系统的根	62	8.2.5 BLP 的各个方面及其局限性	87
6.6.5 挂接文件系统	62	8.3 BLP 的 Multics 阐述	87
6.6.6 环境变量	63	8.3.1 Multics 中的主体和对象	88
6.6.7 搜索路径	63	8.3.2 转换 BLP 策略	88
6.6.8 包裹层	63	8.3.3 检查内核原语	89
6.7 管理问题	64	8.4 深层阅读	90

8.5 练习	90	11.4 数字签名	116
第9章 安全模型	91	11.4.1 一次性签名	117
9.1 Biba 模型	91	11.4.2 ElGamal 签名和 DSA	117
9.1.1 静态完整性级别	91	11.4.3 RSA 签名	118
9.1.2 动态完整性级别	92	11.5 加密	119
9.1.3 调用的策略	92	11.5.1 数据加密标准	119
9.2 中国墙模型	92	11.5.2 块加密器模式	121
9.3 Clark-Wilson 模型	93	11.5.3 RSA 加密	122
9.4 Harrison-Ruzzo-Ullman 模型	95	11.5.4 ElGamal 加密	123
9.5 信息流模型	97	11.6 密码机制的强度	123
9.5.1 熵和平均值	97	11.7 演示	124
9.5.2 基于格的模型	98	11.8 深层阅读	125
9.6 执行监控器	98	11.9 练习	125
9.6.1 执行属性	99	第12章 分布式系统中的认证	126
9.6.2 安全性和活动性	99	12.1 引言	126
9.7 深层阅读	100	12.2 密钥建立和认证	126
9.8 练习	100	12.2.1 远程认证	127
第10章 安全评估	101	12.2.2 密钥建立	128
10.1 引言	101	12.3 密钥建立协议	128
10.2 橘皮书	103	12.3.1 认证密钥交换协议	128
10.3 虹系列	105	12.3.2 Diffie-Hellman 协议	129
10.4 信息技术安全评估标准	106	12.3.3 Needham-Schroeder 协议	130
10.5 联邦标准	106	12.3.4 基于口令的 password-based 协议	131
10.6 共同标准	106	12.4 Kerberos	131
10.6.1 保护配置文件	107	12.4.1 领域	133
10.6.2 评估保证级别(EAL)	107	12.4.2 Kerberos 和 Windows	133
10.6.3 评估方法	108	12.4.3 委派	133
10.7 质量标准	108	12.4.4 撤销	134
10.8 成果是否得到充分利用	108	12.4.5 小结	134
10.9 深层阅读	109	12.5 公钥基础设施	135
10.10 练习	109	12.5.1 证书	135
第11章 密码学	110	12.5.2 证书权威	135
11.1 引言	110	12.5.3 X.509/PKIX 证书	136
11.1.1 旧的范例	110	12.5.4 证书链	137
11.1.2 新的范例	111	12.5.5 撤销	137
11.1.3 密钥	111	12.5.6 电子签名	137
11.1.4 密码机制	112	12.6 可信计算—证明	138
11.2 模运算	112	12.7 深层阅读	139
11.3 完整性检查功能	113	12.8 练习	139
11.3.1 冲突和生日悖论	113	第13章 网络安全	140
11.3.2 操作检测码	114	13.1 引言	140
11.3.3 消息认证码	115	13.1.1 威胁模型	140
11.3.4 安全哈希算法	115		

13.1.2 通信模型	141	14.4.2 虚拟内存系统(VMS)登录	160
13.1.3 TCP 会话劫持	141	14.4.3 finger 漏洞	160
13.1.4 TCP-SYN 洪泛攻击	142	14.4.4 栈溢出	161
13.2 协议设计原则	142	14.4.5 不可执行的栈	161
13.3 IP 安全	143	14.4.6 堆溢出	162
13.3.1 认证报头	143	14.4.7 类型混淆	162
13.3.2 封装安全有效载荷	143	14.4.8 疯狂的电脑黑客	164
13.3.3 安全关联	144	14.4.9 AS/400 机器接口模板	164
13.3.4 因特网密钥交换协议	145	14.5 数据和代码	165
13.3.5 IPsec 策略	146	14.5.1 远程登录漏洞	165
13.3.6 小结	146	14.5.2 脚本	165
13.4 SSL/TLS	146	14.5.3 SQL 插入	166
13.5 域名系统 DNS	149	14.6 竞争条件	166
13.6 防火墙	149	14.7 防御	167
13.6.1 包过滤	149	14.7.1 防止: 硬件	168
13.6.2 状态包过滤器	150	14.7.2 防止: 类型安全	168
13.6.3 电路级代理	150	14.7.3 预防: 更安全的函数	168
13.6.4 应用层代理	150	14.7.4 检测: 代码检查	168
13.6.5 防火墙策略	150	14.7.5 检测: 测试	169
13.6.6 边界网络	151	14.7.6 缓和: 最低权限	170
13.6.7 局限性和问题	151	14.7.7 反应: 紧跟时代步伐	170
13.7 入侵检测	152	14.8 深层阅读	170
13.7.1 漏洞评估	152	14.9 练习	171
13.7.2 误用检测	152	第 15 章 新的访问控制范例	173
13.7.3 异常检测	152	15.1 引言	173
13.7.4 基于网络的入侵检测系统	153	15.1.1 访问控制范例的改变	174
13.7.5 基于主机的入侵检测系统	153	15.1.2 修订的有关访问控制的术语	174
13.7.6 蜜罐	153	15.2 基于代码的访问控制	175
13.8 深层阅读	153	15.2.1 堆栈检查	175
13.9 练习	154	15.2.2 基于历史的访问控制	176
第 14 章 软件安全	155	15.3 Java 安全	177
14.1 引言	155	15.3.1 执行模型	177
14.1.1 安全性和可靠性	155	15.3.2 Java 1 安全模型	178
14.1.2 恶意程序分类	155	15.3.3 Java 2 安全模型	178
14.1.3 黑客	156	15.3.4 字节码校验器	179
14.1.4 环境的改动	156	15.3.5 类加载器	179
14.2 字符和数字	156	15.3.6 策略	179
14.2.1 字符(UTF-8 编码)	156	15.3.7 安全管理器	180
14.2.2 整数溢出	157	15.3.8 小结	180
14.2.3 数组	158	15.4 .NET 安全框架	181
14.3 规范表示	159	15.4.1 通用语言运行库	181
14.4 内存管理	160	15.4.2 基于代码身份的安全	181
14.4.1 缓冲区溢出	160	15.4.3 证据	181

15.4.4 强名称	182	16.4.2 安全绑定更新	194
15.4.5 许可	182	16.4.3 地址所有权	196
15.4.6 安全策略	182	16.5 无线局域网	197
15.4.7 堆栈遍历	182	16.5.1 无线对等加密	197
15.4.8 小结	183	16.5.2 WPA	198
15.5 cookie	183	16.5.3 IEEE 802.11i-WPA2	199
15.6 简单公钥基础设施	184	16.6 蓝牙	199
15.7 信任管理	185	16.7 深层阅读	199
15.8 数字版权管理	186	16.8 练习	200
15.9 深层阅读	186	第 17 章 数据库安全	201
15.10 练习	187	17.1 引言	201
第 16 章 移动	188	17.2 关系数据库	202
16.1 引言	188	17.2.1 数据库的关键字	204
16.2 GSM	188	17.2.2 完整性规则	205
16.2.1 部件	189	17.3 访问控制	205
16.2.2 临时移动用户标识	189	17.3.1 SQL 安全模型	206
16.2.3 加密算法	189	17.3.2 特权的授予和撤销	206
16.2.4 用户身份认证	190	17.3.3 通过视图的访问控制	207
16.2.5 加密	190	17.4 统计数据库的安全	209
16.2.6 基于位置的服务	191	17.4.1 聚集和推断	209
16.2.7 小结	191	17.4.2 跟踪攻击	210
16.3 通用移动通信系统	192	17.4.3 对策	211
16.3.1 假基站攻击	192	17.5 操作系统的完整性	212
16.3.2 加密算法	192	17.6 隐私	213
16.3.3 UMTS 认证和密钥协议	192	17.7 深层阅读	214
16.4 移动 IPv6 的安全性	194	17.8 习题	214
16.4.1 移动 IPv6	194	参考文献	215

第1章 绪 论

一些涉及新的信息技术(IT)的文章中常常以如下评论开头:

对安全的关注是阻止新信息技术使用的主要原因,但这样一来也妨碍了普通用户和公司享受到这些技术可能带来的所有好处。

这类观点见诸于安全方面的学术专著,见诸于试图说服用户相信其提供服务价值的咨询人员,见诸于安全产品销售人员或负责安全计划的政府官员。安全方面的故事在媒体中炒得很热,并且总是带有强烈的动机,例如 IT 巨人(微软)在安全方面的漏洞或者对看不见的危害的担心(例如肆虐于因特网上的病毒、蠕虫)。

一些别有用心的人自然会找出一些原因来夸大我们面临的威胁,而要获取有力的证据来评估问题的严重程度常常也很困难。同时,在另一方面,任何遭到某种蠕虫或病毒攻击的人都能证实,威胁确实存在。的确,开放式的通信网络如因特网、移动电话系统的广泛使用,让数量巨大的用户群暴露在安全的威胁下。因此,IT 专业人员必须要了解这些网络的潜在漏洞、核心保护机制及其局限性。

这本书主要讲述计算机安全。计算机安全最初的关注点是多用户系统。用户必须相互隔离,非授权用户必须被禁止修改系统软件。目前的关注点放在被认为是网络终端系统的计算设备上。许多安全问题都源于这些设备连在网络上这一事实,或多或少都可能受到“非信任节点”的攻击。传统的网络安全服务保护节点间的信号,当信息被安全地送达对方,保护的任务就完成了。我们要讨论的是终端系统接收信息后在处理信息的过程中发生的问题。

在进入此书技术内容的讨论之前,本章将纵览一些在实战中试图实现安全措施时必须说明的重要问题。部署安全措施(以及通常的 IT 技术)是个管理决策的问题,必须有组织、按部就班地实施技术安全措施才能奏效。管理决策应依据对当前风险和威胁的分析来作出。因此,我们将给出安全管理及风险和威胁分析的简短综述。

目标

- 明确本书所讨论计算机安全的范围。
- 给出安全管理的简介。
- 覆盖风险和威胁分析的基础知识。

1.1 攻击与攻击者

第一代移动电话系统投入使用不久,英国皇室成员就发现他们的一些很私人的通话曝光在一些报纸上。这些系统无任何保护地在移动电话和基站间传送通信信号,因此任何人只要使用合适的设备都可以在通话中进行监听。第二代移动电话系统(如 GSM)随后包括了对无线链路进行保护的加密机制。类似地,在因特网上使用信用卡进行购物交易的机密性问题也引起了人们的关注。基本的因特网协议没有提供机密性保护,因此位于客户和商家之间的第三者可以截获一系列信用卡号并在以后用来进行欺诈性购物。Netscape 开发的安全套接字层(SSL)协议就是用来处理这类相当严重的问题。

但是,真正的威胁可能潜藏在别处。扫描因特网上包含信用卡卡号的信息包的通信只是一

种会给攻击者带来低收益的攻击。而攻击位于商家站点上保护措施很差的、有客户信用卡信息的数据库的服务器将给攻击者带来更大的回报。已经有文章证实出现了这类针对商家服务器的攻击事件，其目的不是为获得信用卡卡号就是敲诈勒索商家。

身份盗用 (identity theft)，也就是使用其他人的“身份”(姓名、社会保障号、银号账号等)来获得对某一资源或服务的访问，它利用了这些服务中使用非保密的身份信息来验证请求的内在漏洞。

接受外部用户输入的软件(如因特网浏览器或邮件软件)中的漏洞可能会导致外部各方对设备的控制。攻击者可能会使这些设备上的数据崩溃或者使用这些设备作为进一步攻击第三方的跳板。蠕虫和病毒正是利用网络传播的广播性或漏洞来广泛扩散的，由此产生的大量通信传输会造成网络和终端系统过载。1988年11月的因特网蠕虫是早期记录完好的这类事件的一个案例 (Eichin and Rochlis, 1989)。近几年出现了针对特定目标的拒绝服务攻击 (DoS)。因此，在设计安全协议时，对拒绝服务攻击的恢复性成了一个新的标准。

回到第一个例子，由于移动电话和基站间的通信传送未加保护，攻击者可以截获用来对客户支付进行认证的“秘密”身份标识符。对欺诈实施者来说，主要路口交汇处是一个埋伏等待的好位置。在盗取了这些身份标识后，攻击者便可以制造出克隆的电话来使用，但计费却记在被克隆的电话的用户账户上。据报道，曾经有段时间，在受到严重破坏的网络中，50%以上的电话呼叫都是这种欺骗性的呼叫。

GSM 使用挑战/回应协议进行用户身份的认证，在认证过程中用户身份的秘密信息不进行传送，因此以上这种攻击就起不了作用。然而，这并不意味着支付过程的所有问题都解决了。今天，我们常常看到诱骗不知情的客户回呼由攻击者拥有的有奖资费号码，使用现有支付系统骗取客户钱财的事情。这种攻击是通过使用(或误用)技术系统，而不是通过利用技术系统的缺陷来实现的。其对策应该在使用的人身上去找，如，在回电应答请求前提高警惕，又如，要弄清在合法的系统中进行交易支付时如何获得客户的同意从其账户中划钱。

以上描述的案例中，攻击都来自于外部。“御敌于城门之外”是计算机安全的传统的范例。然而，对攻击来源的统计分析表明，大部分安全事件和大部分危害来自于内部的攻击(联合国，1999)。有一种说法是通过因特网的攻击可能会改变这种状况，但是在电子商务交易或机构组织机构中，内部欺诈一直都引起相当的关注。

据说，安全工程的目的就是把一个攻击的代价提高到一定层次，使攻击的付出超过攻击者获得的价值。这种见识是比较短浅的。因为并非每一个攻击者的动机都是钱的驱使。因人员过剩而被裁掉的员工可能极想对前任雇主进行报复。黑客可能想展示他们的技术专长，并从攻破拦在他们面前的安全机制中获得特别的满足。“恣意破坏的人”可能不计后果地发起攻击。政治激进分子可能会“黑掉”他们不喜欢的网站。

要攻破一个系统所需要的专业技术通常都有一些相似之处。有些情况下，需要结合对系统内部的了解来实现一个成功的攻击。在这方面，社会工程学 (social engineering) 可能比技术手段更重要 (Mitnick and Simon, 2002)。在电话里和计算机操作员争论以获得一个用户账户的口令是个常用的策略。有些攻击需要对技术的深入掌握。还有一些攻击可以自动执行并且可以从网站上下载，以便那些对这些攻击所利用的漏洞或特性知之甚少的脚本小子 (script kiddy) 也能够使用执行。

1.2 安全

软件可能崩溃，通信网络可能中断，硬件可能失效，操作人员可能犯错误。只要这些故障不

和人为的故意操作直接相关,就不能归类为安全问题。意外的故障可能是可靠性(reliability)的问题。操作错误可能是使用性(usability)的问题。安全关注的是人为的故障。这些故障可能不总是为达到一个特定目标的明显意图,但在某种程度上可能是通过一个人做了不应该做的事情而产生的结果。如上所述,这种行为的原因是多种多样的。安全问题的根本原因是人性的本质。

安全从业者知道“安全是一个人为的问题”,因此不能仅仅依靠技术来解决。合法的系统必须通过数据保护和计算机误用法律来定义可接受行为的界限。在组织机构(可以是公司或大学)内的安全职责归根到底还是管理的问题。用户必须执行和遵守其所在组织机构内的安全规则。当然,正确部署和实施技术策略也是整个解决方案的一部分。

1.3 安全管理

保护组织机构的资产是管理的责任。这些资产包含敏感的信息(如产品计划、客户记录或财务数据)和组织机构的IT架构。与此同时,安全措施常常在工作模式上限制组织机构的成员,并可能会诱惑人们去炫耀安全规则。如果安全指令不是来自上级管理机构而是来自组织的其他部门,这种情况尤其可能发生。

强烈建议在落实安全责任时,这些安全措施的制定都要有上级管理部门的支持。首席执行官签名认可的简要策略文件(policy document)可以作为安全管理的基础。这个文件应该人手一册。然后,就可以组织安全意识(security awareness)计划。这并不是要求所有的成员都成为安全问题专家,但是他们都应该知道:

- 为什么安全问题对于他们和组织机构(公司)如此重要。
- 对每个成员有什么要求。
- 他们应该遵守什么样的好习惯。

用户会忽略显然不合理的安全规则,这其实和安全专家会把不讲理的用户视作敌人一样。强迫用户遵守他们认为武断的安全规则并不是一个有效的方法。研究表明,让用户像股东那样参与到他们组织机构安全机制建设中,可以让他们自觉地遵守安全规则而不是抵触安全规则(Adams and Sasse, 1999)。

一个公司开发IT服务或产品时,也要给其开发人员提供安全培训。与安全相关的系统部分与其他系统部分之间并没有很清楚的界限。所以让开发人员了解开发的服务部署的环境、以及预期的危险,有助于使他们重视保护系统安全的需要,即使他们并没有亲自实现保护机制。开发人员也要警惕一些敏感数据的分类。例如私人数据,需要根据特定的规则和规章处理。最后,开发人员应该及时知道编程中易被攻击的编码漏洞。

1.3.1 安全策略

安全策略说明了应该保护什么同时也可能会表明该怎样保护。为了不引起混淆,我们将遵照Sterne(1991)中给出的安全策略的定义,并区分有组织的和自动的安全策略之间的区别。一个策略有其特定的目标:

安全策略目标 即是一个声明,用于保护可识别资源,防止未授权使用。

同时安全策略也应该解释该目的应该怎样达到。这一点应该先在组织一级实现。

有组织的安全策略 一组法律、规则和实践的集合。这个集合用来规范一个组织的管理,保护和分配资源以达到特定的安全策略目标。

在IT系统中,有组织的安全策略可以由技术方法支持。

自动的安全策略 一组限制和属性的集合。这个集合用来指明一个计算系统怎样保护信息

和计算资源不被非法地使用而违背有组织的安全策略。

自动的安全策略解决这样一些问题对访问控制列表或者对防火墙设置的定义,对可能运行在设备上的服务和用于保护网络通信的安全协议的决策。

1.3.2 测量安全

测量安全是安全工程的关键。为了让经理(或者客户)相信新的安全机制带来的好处,如果我们能够测量引入安全机制前后系统的安全程度的话不是很棒吗?确实,没有前面测量得出的定量数据,想要作出一个良好的管理决策是很难的。理想情况下,测量(measurement)可以得出一个定量结果,和别的测量结果做比较,而不仅仅是对被分析的产品或系统安全的定性声明。

- 产品(product)是指 IT 软件、固件和/或硬件的包,它可提供专为多种系统的使用或合并而设计的功能。
- 系统(system)是特定的 IT 装置,有特定的目标和操作环境(CCIB, 2004a)。

产品测量可以表明它的安全潜力,但是即使是安全的产品也能被不安全地部署。一个显著的例子就是服务账号的默认口令不能更改。因此安全管理的任务是确保提供的安全属性被正确地使用。对于产品,可以用其检测到的安全缺陷(漏洞)数作为安全度的衡量依据。根据追踪发现缺陷的时间可以作为预测下一个缺陷发现时间的基础。相关的一套方法已经在软件可靠性(software reliability)领域完善起来。这套研究方法假定缺陷的检测和漏洞代码的调用由一个给定了验前的概率分布函数或者一组给定的还要估计参数的分布函数控制。另一个建议是测量一个产品的攻击表面(attack surface),如对外调用的接口数量和和代码中危险命令的数量。

这些建议在某种意义上来说给出了定量的结果,但它们是否真正测量了安全程度还是有争议的。安全漏洞的数量究竟有多大关系呢?攻击者往往很容易找到和利用一个漏洞来威胁系统的安全性。考虑到很少有两种产品都完全相同地解决一个问题,用这种定量的方法作为有意义的比较产品安全性的基础还是有争议的。因此,建议这些测量的数据作为定性的测量用于监控一种产品的优化。

对系统的安全测量可以参考部署的实际配置。在有访问控制特性的系统中,我们可以参考有系统优先权的账户数量或者参考弱口令的账户数。在一个联机系统中,我们可以参考开放的端口数和从外部可访问的服务,或者现在运行的版本是否有已知的易被攻击的地方。这些属性肯定能提供有价值的状态信息(status information),但是并不会真正给出测量要求的定量的结果。

对于计算机网络,我们可以测量网络中节点的连通性,以评估攻击能传播多快多远。我们也能测量在攻击之后服务瘫痪的时间,或者在知道配置的情况和攻击的程度时预测修复恢复服务所要花费的时间和费用。

另外一种方法是通过测量开始攻击的花费来测量安全性。要考虑:

- 在攻击中,攻击者需要投入的时间,例如分析软件产品以找出漏洞。
- 攻击者的花费,例如计算周期或特殊的装备。
- 发动攻击必要的知识储备。

然而,在第一时间发现攻击所要的花费经常要比发动攻击需要的花费多得多。今天,因为有了攻击脚本(attack script)使得只要一点投入或者只要知道一点所要攻击系统的漏洞就可以发动攻击。

还有一个方法,我们可以关注给定系统的资产,并测量这些资产被暴露的风险。1.4.4 节将给出风险和威胁分析的概论。总之,理想的安全策略是,我们能对一些安全的个别方面有最好的度量,搜寻更好的度量仍然是一个开放的研究领域。

1.3.3 标准

已制定的安全管理标准规定了在特定的工业分支中需要采用哪些安全措施。典型的例子是金融行业的规定^①，或者是政府部门中处理机密材料时的规定^②。

其他管理标准应该描述为安全管理的最佳实践的代码。这些标准中最著名的是 ISO 17799 (国际标准化组织, 2001)。它不是安全产品, 也不是对产品和系统的评价标准集合的技术标准。ISO 17799 的主要内容如下。

- 建立有组织的安全策略: 这个文件对安全问题提供了管理指导和支持。
- 组织性安全基础设施: 在企业中, 应很合理的组织的安全责任。管理人员应该可以得到企业精确的安全状态。报告制度应该能促进有效的交流和安全决策的实现。当信息服务外包给第三方时, 安全性应该被保持。
- 资产分类和控制: 要确定什么值得保护, 要投入多少开销保护。企业应该有自己的清晰的资产及其价值的资料。
- 物理和环境安全: 物理安全措施(保卫、门锁等), 保护进入商业重地和敏感区域。例如只有被授权的人可以进入服务器所在地。这些措施可以防止非法访问敏感信息和盗取设备。自然灾害发生的可能性取决于环境因素, 例如该地区是否易遭洪水?
- 个人安全: 你自身或者承包者可能是不安全来源。对于新雇员的加入和雇员的离开都应该履行一些手续和程序(例如收回钥匙和准入证, 删除离开的雇员账户)。加强假期管理可以阻止雇员隐藏他们作假的痕迹。对于新雇员做背景核对也是个不错的主意。某些部门中这些检查是法律所要求的, 但也有隐私法来限制雇主可以调查雇员的哪些信息。
- 交流和运作管理: 日常的 IT 系统管理和商业处理必须保证保持了安全性。
- 访问控制: 对数据、服务和计算机实行访问控制。应特别注意对远程访问的控制。例如通过因特网或者拨号连接。自动的安全策略定义了应该怎样实施控制访问。
- 系统开发和维护: 一个信息系统在开发过程中应该考虑到安全问题。运行安全取决于恰当的维护(例如, 漏洞的修复、更新病毒扫描)。信息系统支持部分的引入应当是安全的(怎样处理忘记口令的用户), 同时 IT 工程的管理应该考虑到安全问题(谁来写敏感的程序, 谁能够访问敏感的数据)。
- 商业持续规划: 制订好措施以应付主要的故障和事故。首要的措施是将重要数据的备份转移至不同的地点, 接着可能继而向它们提供远程备用计算设备。同时你还需要为失去骨干人员做好准备。
- 规范性: 公司既要承担合法的、符合规章和契约规定的职责, 也要遵守职业规范和他们自己的有组织的安全策略。当尝试最小化审计过程对商业过程的干扰时, 应保证审计过程的有效性。在实际工作中, 这些方面通常会是一个比技术上的安全测量更巨大的挑战。

达到 ISO 17799 的标准是一项繁重的任务。公司目前的状态相对于标准应当是确定的, 任何已知缺点应该得到解决。已经存在这样的软件工具, 可以自动处理部分的这类过程, 最好的应用就是实践, 只有此时才能确保与标准的一致性。

1.4 风险与威胁分析

工程和商业的许多领域都发展了自己的风险分析规范和术语。在本节中, 我们将概述信息安

① 比如 Visa 支持的信用卡行业数据安全标准。

② 比如美国策略声称他们使用 192 比特或 256 比特加密算法 AES 可以用于绝密数据的处理。

全中的风险分析。在 IT 安全中，风险分析通常应用于如下几个方面：

- 企业全面的信息资产。
- 企业特殊的信息基础结构。
- 产品和系统开发过程，例如，软件安全领域。

通常来说，风险是一些偶然事件或攻击对企业造成损失的可能性。对信息技术系统的攻击由一系列利用系统漏洞的操作组成，直至攻击者的目标达成。评估由攻击形成的风险时，我们应当估算攻击造成损坏的数量和这种攻击事件的可能性。这种可能性取决于攻击者的动机和这种攻击所需支持的难易程度。话说回来，这进一步取决于遭受攻击的系统的配置。

在风险分析过程中，为了解开需要追踪研究的多方面的问题，我们将涉及到资产、漏洞和威胁，并将它们用到计算风险分析的函数中。非正式的计算公式为：

$$\text{风险} = \text{资产} \times \text{威胁} \times \text{漏洞}$$

在风险分析的过程中，结果取决于资产、漏洞和威胁。在定量的风险分析中，结果取决于数学定义域，就像一个概率空间。例如，给资产赋予货币值，威胁赋予几率值，那么预期的损失就可以计算出来。在定性的风险分析中，结果由那些不具基础数学结构的域给出。风险的计算基于整合了的安全专家建议的规则给出。

1.4.1 资产

首先，资产应当是被鉴定和评估过的。在一个 IT 系统中，资产包括以下几个方面。

- 硬件：膝上电脑、服务器、路由器、个人数字助理(PDA)、移动电话、智能卡等。
- 软件：应用程序、操作系统、数据库管理系统、源代码、目标代码等。
- 数据和信息：运行和规划事务的必需数据、设计文档、数字编目、客户信息等。
- 名誉。

资产的鉴定需要通过相关的直接系统的训练。资产的估价不仅仅是一项挑战。一些资产，例如硬件，可以用它们的等价货币值来估价。但是，另外一些资产，例如数据和信息的评估会更加困难。如果商业计划泄露给了竞争对手，或者客户的私人信息被泄露，你应当计算因丢失商业机会而造成的间接损失。你的对手可能会比你叫价低，你的客户也可能会抛弃你。甚至设备遗失或被盗窃时，你必须考虑到上面存储的数据的价值和运营在设备上的服务的价值。在这种情况下，资产的价值根据它们的重要性而决定。判断重要性的一个手段是，想想如果某一资产遭受损坏，你的生意能维持多久：一天、一星期还是一个月？

1.4.2 漏洞

漏洞是系统的弱点，会偶然地或蓄意的被人利用致使资产受到破坏。在信息技术系统中，典型的漏洞包括：

- 系统特权账户没有更改默认的口令，例如“MANAGER”。
- 程序有多余的特权。
- 程序存在已知缺陷。
- 资源的弱访问控制，例如，内核的完全可写。
- 弱防火墙配置，允许对有漏洞的服务的访问。

漏洞扫描仪(vulnerability scanner)提供了系统的、自动漏洞识别功能。其已知漏洞的信息知识库需要不断保持更新。像 SANS 或者计算机紧急响应组(CERT)这样的组织就提供此类信息，同时给软件公司提供安全建议。

漏洞可由它们造成的影响定级(危险级别)。让攻击者获得系统特权账户的漏洞比获得底层用户账户的漏洞具有更高的危险等级。允许攻击者完全模拟用户的漏洞比让用户只能在特定环境中模拟单独的特殊服务的漏洞拥有更高的危险等级。一些扫描工具会在侦测到漏洞的同时为其标出等级。

信息技术安全中的术语出了名的不精确,你可能会发现漏洞扫描仪被当作是风险分析工具来出售。这无疑有些挂羊头卖狗肉的嫌疑,所以你的职责就是找出这些所谓的“风险分析工具”事实上能提供什么,然后将它物尽其用。

1.4.3 威胁

威胁是对手利用漏洞去损坏资产的行为。识别威胁的方法各异。我们可以根据对资产造成的损坏对威胁进行分类。例如,微软有关软件安全的 STRIDE 威胁模型分类列表如下。

- 欺骗身份认证:攻击者伪装成某用户。
- 数据篡改:例如,安全配置被更改以便攻击者能获得更多的特权。
- 抵赖:用户否认执行了带有攻击性的操作或者完成了交易。
- 信息泄露:信息因为泄露给了错误的对象而可能会丧失原有的价值(例如商业秘密);机构可能因没有恰当地保护信息而面临处罚(例如私人信息)。
- 拒绝服务攻击(DoS):DoS 攻击会造成站点的临时不可用;已经有相关出版物讲述了利用这种攻击来损害竞争者的事件。
- 提高特权:用户在计算机系统上获得比自己原有资格更高的特权。

接下来我们可以鉴别攻击的来源。对手是机构的一员、外部人员、订约人或者以前的成员?对手是直接访问你的系统或是进行远程攻击?

我们同样可以分析攻击执行的细节。一开始,攻击可能无害,它只是收集必需的信息以获取机器上的特权,再转到另一机器上,直到达成最后的目的。为了得到更加完善的潜在威胁图,可以构建攻击树(attack tree)。攻击树的根是一般类的攻击。树节点是达成攻击所需的子目标。子目标可以进一步划分为子目标。其中有 AND 和 OR 节点。为了到达 AND 节点,所有的子节点都必须达成。OR 节点只要一个及以上的子目标达成即可。图 1-1 给出了基本的口令获取攻击树。口令可通过猜测、哄骗操作员说出,或者监视他人而得到。猜测分为在线和离线。对于离线猜测,攻击者需要获得加密口令并完成字典攻击。攻击者也可以人工监视受害者(即所谓的 shoulder surfing),在键盘上安装摄像机或话筒,通过声音区分所击的键。

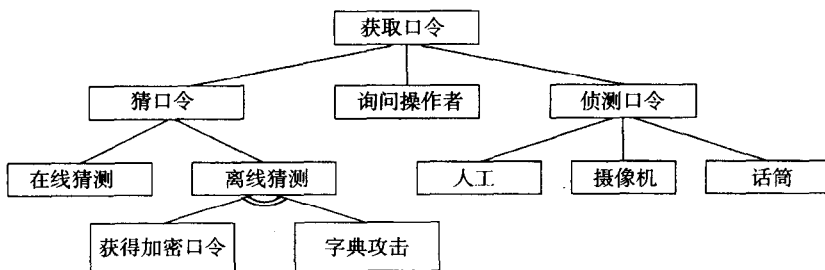


图 1-1 获取用户口令的攻击树

可以为攻击树的边赋值。这些值可以估算出攻击的成本、发生的可能性、成功的可能性,或其他方面。通过这些值,我们可以计算最廉价的攻击、最有可能发生的攻击或者最可能成功的攻击。

攻击树是一种形象化、结构化的威胁分析方法。当全局的威胁评估可以由独立子目标判断时，威胁评估便具备了可复制性。如果最后的结果难以置信，那么应当考虑树的哪些子目标对最后的结果起关键的作用，那些值可能应该调整为更“合理”的值。以上要点解释了为什么构造攻击树不仅是科学也是一门艺术。你需要经验来告诉自己应该什么时候重新调整子目标的等级，什么时候调整对某个威胁的严重性的预期看法。你同样需要经验来知道应该什么时候停止子目标的继续划分，即交易中的一种称作分析瘫痪 (analysis paralysis) 的现象。

威胁可以根据它们的可能性来评定。可能性决定于攻击的难易程度、攻击者的动机以及潜在的攻击者人数。攻击脚本使攻击变得自动化，发动攻击更加容易。它们还可能为大量攻击者所适用。因此，这种攻击与个别的手工攻击比较起来更有可能性。

1.4.4 风险

在评定资产价值、漏洞危险程度和威胁可能性之后，我们开始复杂的风险计算工作。

定量风险分析中，预期损失可以按照概率论中的构架来计算，它是基于资产货币价值和威胁可能性的概率。这种方法不仅拥有基于良好根基的数学理论的优点，同时也有相当的缺陷，即我们所得到的评定往往是根据事实或经验作出的猜测。简而言之，我们所获得的结果的质量不如提供的投入的质量高。我们可以考虑使用其他的数学构架比如模糊理论来防备评定的不精确性。某些风险分析领域可进行定量分析，但更多的是由于投入的不精确性导致无法确定采用何种数字处理方式。

定性风险分析如下。

- 资产可根据其重要等级来评定：很重要—重要—不重要。
- 漏洞的危险程度可根据如下标准来评定：立刻修补—尽快修补—呆会修补—方便时再修补。
- 威胁可根据其可能性大小来评定：很可能—可能—不可能—极不可能。

可提供一个好的衡量粒度，比如从 1 ~ 10 的数值。无论用什么方案，都必须给予评定的指南。资产评定、漏洞和导致风险的威胁的映射常由拟定的反映保密专家的判断的表给出。补充 STRIDE 的 DREAD 方法论可作为定性分析的一个方案 (Howard and LeBlanc, 2002)。

- 潜在损害：与受损资产价值相关。
- 可复制性：发动攻击难易的表现；可再生的攻击比只在特定环境下才有效的攻击的风险更大。
- 可利用性：与发动攻击所需的努力、技术和资源相关。
- 受害用户：对软件销售商来说，这是另外一个会导致潜在损害的重要因素。
- 可发现力：攻击何时被检测到？在严重破坏的情况下，无法获知系统已经中招了（在第二次世界大战中，德国的情报系统就拒不相信他们的很多加密方案已被多次破解）。

1.4.5 对策——减轻风险

风险分析的结果是：一份按优先级排序的威胁列表以及一些建议的用于减轻风险的对策。风险分析工具通常与应付威胁的对策的知识一起运用。

在决定使用哪种安全措施之前先进行风险分析，这种做法看似很对。但是，有两个原因可以说明为什么这个想法不可行。为一个大型机构执行风险分析需要一定的时间，但是该机构的 IT 系统和其周围的环境却在不断变化。所以，到给出分析结果时，有些分析已经过时了。此外，管理部门很难界定一份全面的风险分析的开销。

由于这些原因,组织机构或许会选择基准防护(baseline protection)作为折中的方案。这种方法分析了典型案例的安全需要并推荐适度的安全。德国信息安全局就一直在维护一个很有名的IT安全性基准文件(BSI, 2003)。

1.5 深层阅读

Anderson 关于安全工程的书全面阐述了安全性面临的难题并给出了出色的见解(2001)。关于安全管理和一般意义上的IT安全性问题的有效讨论可以在Smith(1993)的书中找到。有关安全策略的各种术语的含义在Sterne(1991)的书已给出。Martin Smith(1993)关于在商业机构中定义安全策略的讨论直到今天仍然有意义。信息安全风险管理在Alberts and Dorofee(2003)中有所论述。

1.6 练习

练习 1.1 讨论衡量产品和系统安全的高级选择。

练习 1.2 在你使用的计算系统中,识别出潜在的、可能包括安全机制的软件组件。

练习 1.3 即使计算机中心处于地势高且干旱的地势,危险分析中是否也应该包括洪水对计算设备的破坏?

练习 1.4 对一个手机服务进行风险和威胁分析,可在分析中考虑消息是通过手机与基站之间的无线电发送的,以及国际漫游用户在国外旅行时可以使用所谓的访问网络中的服务。分析时,试思考用户和网络操作员的观点。

练习 1.5 银行客户可以在自动取款机(ATM)上使用现金卡和个人识别码(PIN)取出现金。同时从客户和银行两个角度来考虑该应用的风险和威胁分析。

练习 1.6 考虑一个大学某系的中心服务器失窃的情况。如果发生这种情况,哪些资产可能被破坏?试对此威胁构建一棵攻击树。

第2章 计算机安全基础

如果没有明确定义“计算机安全”这个主题的含义和范畴，我们就不能有意义地探究这个主题。如果没有一些一般性的指导原则来帮助我们理解今天将会遇到的多种概念和安全机制，我们也不可能开始这样一个讨论。因此，我们的首要任务就是搜寻一个计算机安全的定义。为了避免孤立地讨论个人的安全系统，我们将会提出一组一般性的能够指导安全信息处理系统设计的工程学原则。非常欢迎读者在整本书的多种安全系统中找出这些原则。

目标

- 找到一个计算机安全的定义，介绍机密性、完整性和可用性。
- 解释计算机安全在基本层面上的进退两难的困境。
- 提及一些在构件安全系统时必须制定的一般性的设计决策。
- 指出计算机安全机制必须依靠物理的或者有组织的保护措施才能够生效。

2.1 定义

按照良好的学术传统，应通过定义研究目标来开始研究工作。至少，我们应该尝试这样做。计算机安全研究是在计算机系统中维护安全所使用的技术。不必去区分计算机系统（不确切地说，是指内有处理器和存储器的装置）和信息技术（IT）系统（不确切地说，是指紧密耦合的计算机系统网络），因为随着技术的不断飞速发展，现代的计算机已经是一种由各部件紧密耦合而成的网络，曾经是应用程序的软件也可能成为操作系统的一部分。Web 浏览器就是当前这种趋势的一个鲜明例子。在机器上运行的软件不需要存储在机器上，可以来自于因特网上的 Web 服务器。由此，可以将“计算机安全”和“IT 安全”作为同义词来使用，而不会引起太多的误解。

乍一看，安全好像是一个显而易见的概念，然而，当试图讲述它的确切含义时，似乎离显而易见越来越远了。人们在起草计算机安全的定义以及随后对这些定义进行修正时已经做了大量的努力。但是这些文档的编写者几乎无一例外地都被批评为眼光过于狭窄，或过多地介入了严格意义上来说不属于计算机安全的一些计算机科学领域。

2.1.1 安全

安全是关于资产的保护。这个定义意味着必须知道资产及其价值，这种带有一般性的看法当然也适用于计算机安全，在此书的 1.4 节中已经提到风险分析的作用。本章主要关注计算机系统的保护措施，对这些保护措施的大致分类如下。

- 预防：采取措施防止资产遭受损害。
- 检测：采取措施检测出什么时候资产受损，它是怎么受损的，以及是谁造成的损害。
- 反应：采取措施以便能够重新获得资产，或者使资产从损害中恢复。

为了说明这一点，可以你家中贵重物品的保护为例进行说明。

- 预防：门窗上的锁以及窗门使盗贼难以闯入你家；保护地产的围墙，或中世纪城堡外的护城河可增加另一层保护。
- 检测：当某样东西不在某个地方时，你可能会察觉到它被偷了；当入室事件发生时，报警

器便会鸣响；闭路电视提供的信息可以帮助确定入侵者。

- 反应：你可以打电话报警。你可能决定用其他物品替代被偷物品，警察可能会追回失窃物品并送还给你。

来自现实世界的例子有助于解释计算机安全的原则。然而，将物理安全与计算机安全进行类比并不总是可能的或是可取的。事实上，有些术语应用于信息技术环境时很容易引起误解。为此列举与讨论的领域有密切关系的例子，考虑使用信用卡号在因特网上购物的情况。一个骗子可能使用你的信用卡号购物，而费用从你的卡上扣除，你该怎么来保护你自己呢？

- 预防：下订单时使用口令；依靠贸易商行在接受信用卡购物订单前对购物者进行某些检查；不在因特网上使用信用卡号。
- 检测：一项未经你授权的交易出现在你的信用卡账单上。
- 反应：你可以申请一个新的卡号，欺诈交易的损失可以由持卡人、骗子购物的商行或者信用卡的发行商承担。

在这个例子中，骗子“窃取”了你的卡号，但你依然持有这张卡，这和你卡被盗的情况是不同的。因些在某些法律框架下，比如在英国，骗子不能被起诉为盗窃你的信用卡卡号。必须制定新的律法来解决这个新的问题。

考虑保护机密信息的选项，继续关于保护问题进行讨论。有可能仅当秘密揭露时才发现已经泄密，而在某些情况下，损失已无法挽回。你的竞争对手可能已经得到了你耗费数年功夫完成的产品设计，在你之前占领了市场，并获取了全部的利润，而你却歇业了，在这种情形下，预防是唯一的选择。这也解释了为什么历来计算机安全相当重视对机密信息泄露的预防。

在预防和检测之间并不总是存在折衷方案。实践证明，你投入到预防中的财力和物力越多，那么为了确认预防确实有效，你也就不必在检测中投入越来越多的财力和物力。

2.1.2 计算机安全

为了获得计算机安全概念，可先分析信息资产是如何受到损害的。最常见的定义包括以下三个方面。

- 机密性 (confidentiality)：防止未经授权的信息泄露。
- 完整性 (integrity)：防止未经授权的信息被篡改。
- 可用性 (availability)：防止未经授权的信息或资源被截留。

读者可以立即就这些话题的优先顺序展开讨论，并且给出一个重新排序的例子。或者读者可以论证说这个列表不完整——因为列表永远不会完整——并再增加一些内容，比如读者关心通信，可以增加真实性 (authenticity)；若对电子商务类应用感兴趣，则可以增加问责性 (accountability) 和不可否认性 (nonrepudiation)。

即使在这一般的层次上，仍会发现对于某些安全方面的精确定义有不同意见。因此，通常会给出定义的出处，这样定义的背景就比较清楚了。我们将选择一些对于计算机安全发展非常重要的文档，像 US DoD Trusted Computer System Evaluation Criteria (橘皮书；美国国防部，1987)，European Information Technology Security Evaluation Criteria (ITSEC；欧洲共同体委员会，1991)——第10章包含了以上两部分内容，还有 International Standard ISO 7498-2 (国际标准化组织，1989)，针对通信安全的 ISO/OSI 安全体系结构 (现在已被 ISO 10181 取代，但是仍然非常具有影响力)。上面的定义就出自 ITSEC。

2.1.3 机密性

历史上，安全和保密是紧密联系在一起的。即使在今天，许多人仍然觉得计算机安全的主要

目的是阻止未授权的用户阅读敏感信息。通俗地说,未授权用户不应该获悉(learn)敏感信息。机密性(隐私、秘密)就是要解决计算机安全这一方面的问题。术语“隐私”(privacy)和“秘密”(secrecy)有时候被用来区分个人数据的保护(隐私)和机构数据的保护(秘密)。机密性概念已经被很好地定义了,并且计算机安全中的研究通常集中在这个题目上,因为最起码它提出了与物理安全不相类似的新问题。有时候,安全性(security)和机密性甚至被用做同义词。

一旦研究深入机密性时,将会面对这样一个问题,即对未授权用户隐藏文档内容还是干脆将存在这一事实隐藏起来。为了了解采取这一额外步骤的原因,可考虑在通信系统中的流量分析。对方仅查看到谁在什么时间与谁连接,而不查看传送的信息内容。即使如此,观察者从通信双方的关系中仍然可以得到有用的信息。在流量分析中,可能需要某些事件的不可连接性(unlinkability)。如果想要隐藏谁在从事某个活动,可以要求一定的权限,比如匿名(anonymity)权限。

在纸质文档的世界中,控制对文档的访问很简单,只要列出那些允许阅读这个文档的人就可以了。不同的是,计算机安全为实现机密性必须控制写操作,读者在8.2节将了解更多有关这方面的内容。

2.1.4 完整性

很难给出一个简洁的完整性定义。一般来说,完整性是要确保每个事物保持它该有的状态(很抱歉给出这样一个毫无帮助但却反映了现实的定义)。在计算机安全的范围内,我们可能会满足于2.1.2节中引用的一个定义,即完整性研究如何预防未授权的写操作。当这种解释和信息流程策略(参见第9章)一起使用时,完整性是双重机密性,我们可以期望用类似的技术同时达到两个目的。

然而,更进一步的问题像“被授权做某人做的事”和“遵循正确的程序”也已经被包含在完整性的定义中。*Clark and Wilson*(1987)在他们的颇有影响力的论文中采用了这个方法,该论文声称完整性就是这样一种属性:

系统中的任何用户,即使被授权,也不允许以一种会使得公司的资产或账目记录丢失或被破坏的方式来修改数据项。

如果将完整性与预防所有未授权的行为等同起来,那么机密性就成为了完整性的一部分。

迄今为止,我们都是通过阐明用户必须控制的行为来获得安全性。而在定义完整性时,从系统的观点来看最好把注意力集中在系统状态上。橘皮书(美国国防部,1985)中关于数据完整性的定义正是这种类型的。

数据完整性(data integrity) 当计算机处理的数据与源文件中的数据一样且没有受到意外的或恶意的修改或破坏时所呈现的状态。

在这里,完整性是外部一致性(external consistency)的同义词,存储在计算机系统外的数据应当正确地反映计算机系统外的某些事实,这当然是非常理想的情况,然而,仅靠计算机系统内部的机制来保证这个特性是不可能的。

更为混乱的是,信息安全的其他领域有它们自己的完整性的概念。比如,在通信安全中,完整性是指:对因蓄意(intentional)操作或随机的传输错误而引起的传输数据的修改、插入、删除或重放而进行的检测(detection)和纠正(correction)。当没有人被授权进行修改时,可以把蓄意的修改看作是未授权修改的一种特殊情形。但是,采取这种观点并没有多大帮助。因为授权体系的有或无对于需要解决的问题的本质以及相应的安全机制都有影响。

完整性通常是其他安全属性的先决条件。比如,攻击者可能试图通过修改操作系统或操作

系统引用的访问控制列表来挫败机密性控制。因此,我们必须通过保护操作系统的完整性或者访问控制列表的完整性来实现机密性。

最后,应当注意的是还有更常见的完整性定义,它们把安全性和可用性都当作是完整性的一部分。

2.1.5 可用性

ISO 7498-2(国际标准化组织,1989)中给出的定义如下:

可用性(availability) 需要时可被一个授权实体访问和使用的属性。

可用性是超出传统计算机安全范围的一个非常令人关注的问题,用于改善可用性的工程技术通常来自于其他领域,例如容错计算。在安全领域,希望保证的是,恶意的攻击者不能阻止合法用户合法访问他们的系统,也就是说,希望阻止拒绝服务攻击(denial of service)。对此,再次引用 ISO 7498-2 中的定义:

拒绝服务攻击(denial of service) 阻止对资源的授权访问或者推迟时间关键的操作。

近来,因特网上已经出现了洪泛攻击(flooding attack)事件,攻击者用大量的连接请求攻击服务器,致使服务器瘫痪。图 2-1 讲述其中的第一式拒绝服务攻击(Smurf 攻击)。攻击者以假冒的发件人地址(受害者的地址)发送一个 ICMP 的回应请求到一些网络的广播地址。回应请求将分发到所有的网络节点中。每个节点均会对假冒的发件人地址进行回复,从而使受害者被回复包淹没。这种广播地址提供的扩大攻击对攻击者相当有利。

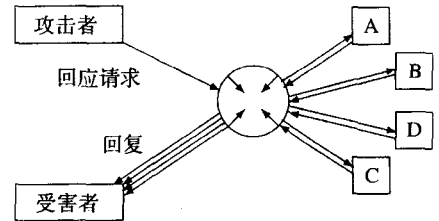


图 2-1 拒绝服务攻击(smurf)

在许多情形下,可用性都可能对计算机安全最重要的特性,但处理这类问题的安全机制却是特别匮乏。事实上,限制太多的或太昂贵的安全机制本身就可能导致拒绝服务。安全协议的设计者现在通常会尝试避免负载的不平衡,因为这种不平衡常常会允许恶意一方以自身较小代价超载通信。

2.1.6 问责性

现在已经讨论了计算机的安全的三个传统领域,回想一下,可以看到它们都研究了访问控制的不同方面,并且把重点放在了预防(prevention)令人讨厌的事件上面。必须接受这样的事实,即几乎从来都无法防止所有不正确的行为。首先,可能会发现授权行为可以导致不安全;其次,可能会在我们的安全系统中发现一个缺陷,使得攻击者可以绕过控制。因此,需要在安全列表中加入新的安全要求:用户应当为自己的行为负责。这个要求在刚出现的电子商务世界中尤其重要,已收入类似橘皮书这样的历史性文档中:

问责性(accountability) 审计信息必须有选择地保存和保护,以便影响安全的行为可以被追溯到责任方。

为了做到这一点,系统必须识别(identify)和认证(authenticate)用户,必须保存一份安全相关事件的审计记录,如果发生了违背安全事件,审计记录的信息可以帮助确认作恶者,以及针对作恶者损害系统所采取的步骤。

2.1.7 不可否认性

不可否认服务能对发生的具体行为提供不可欺骗的证据(unforgeable evidence)。在分析加密机制提供的安全服务时,这种定义相当有意义。数字签名(digital signature;参见11.4节)提供了不可否认性,其实就是一些能被第三方证实的证据。通信安全中典型的不可否认服务有两种:发送方不可否认(nonrepudiation of origin)和递送方不可否认(nonrepudiation of delivery)^①。发送方不可否认提供了关于文档发送者的证据,递送方不可否认提供了关于消息被发送给具体的接收者的证据。

由于不精确的语言问题,对不可否认服务的讨论从来就未曾中断过。当邮件被发送到邮箱时是否应该谈论接收方不可否认呢?甚至还有更多的关于不可否认服务的常识性误解。有时候,人们称不可否认服务为一些事件提供了“确凿的证据”,这些证据可以“被世界上的任何一个法庭接受”。认为数学证据能够使某个人不得不承认卷入了一件有争议的事件中的想法是相当天真的。确凿的证据这样的概念与很多法律系统都格格不入。我们会在12.5.6节中简要地探讨这个话题。

2.1.8 可靠性

在进行安全性讨论时,还必须提及计算的其他领域,像可靠性(涉及“意外”故障)和安全性(safety,涉及系统故障对环境的影响,也涉及到系统在不利条件下必须正确运转的情形)。提及这些领域的首要原因是概念上重叠,依你的观点而定,安全性可以是可靠性的一个方面,或者反之亦然。IFIP WG 10.4试图避免这种进退两难的处境,它引入可信任性作为一个统一的概念,并将安全性、可靠性、完整性和可用性当作可信任性的几个方面。

可信任性(dependability) 计算机系统的一种能使系统提供服务有理由地被信任的特性。

这里提及的系统提供的服务就是它的行为,用户是与系统交互的另一个系统(人类)。

原因之二是有些应用必须同时解决多个问题。假设有一个处于安全应急应用中的计算机系统,在一些突发事件中,该系统用户必须对紧急事件做出反应。安全控制则应当阻止入侵者制造恶性事故,并通过搜寻陌生的行为模式来识别入侵。对紧急事件的反应有时很陌生——但愿紧急事件很少发生,因此入侵检测可能会将紧急情况下的合法行为误解为攻击,并启动安全机制,这可能会把问题搞糟,因为安全机制会干扰应急人员的行为。一般来说,你不能孤立地追求安全而不考虑你想要保护的系统的其他要求。

最后是因为应用于这两个领域的工程方法是类似的,比如,评价安全软件的标准和评价安全应急软件的标准有许多相似之外,有些安全专家希望最终将只有一个标准。

2.1.9 计算机安全定义

在本书中,我们将采用以下的计算机安全定义:

计算机安全(Computer Security) 研究如何预防和检测计算机系统用户的非授权行为。

在该定义中,正确授权和访问控制这两个概念是计算机安全的基本组成要素。正确授权假设存在一个安全策略(security policy)——在1.3.1节中已解释过。我们可以将纠正不正确行为的影响包括到安全定义中,但这个方面对于进一步的讨论作用不大。

① 除了以上提到的两种典型的不可否认服务,还有一种常见的不可否认服务——接收方不可否认(nonrepudiation of receipt)。

此定义描述了我们在计算机安全方面所能做的工作。当回顾我们为什么要维护计算机安全的最根本原因时，我们可以采用这样一个定义：

计算机安全 与我们采取的用来处理某些不受欢迎的人的蓄意行为的措施有关的一切。

这个定义并没有提及未经授权行为，也没有涉及攻击。但是使得计算机安全的界限更宽，甚至包括了像垃圾邮件这样的问题。自主发送的邮件不一定是攻击。（如果你正在找工作，你可能会把你的 CV 发给一些可能会觉得你的技能很有用的公司，即使没有空缺。）相似的，接收一封自主发送的邮件也不一定是件不受欢迎的事情。这会根据一个人每天收到的邮件的数量和本质而改变。当邮件的滥用变得很讨厌时，反垃圾邮件就成为了一个安全问题。

经验教训

对术语的初步讨论得出的主要结论如下：

- (1) 不存在单一的安全定义。
- (2) 当你阅读文档时，注意不要将你自己的安全概念与文档中使用的概念相混淆。
- (3) 大量的时间都花费（和浪费）在试图定义明确的安全概念上。

2.2 计算机安全进退两难的困境

随着依赖计算机安全的用户数量从处理机密数据的少数公司发展与因特网相连的每个人，人们对计算机安全的需求已经发生了巨大的变化。至少，这种变化已经导致了一种基本的进退两难困境的出现：

不懂安全的用户有特殊的安全需求，却常常不具备安全的专门知识。

这种两难处境在当前的安全评估（security evaluation）策略中显而易见。通俗地说，安全评估会检查产品是否提供了它所承诺的服务。因此，必须规定安全系统的功能，而且我们也需要确保安全控制是有效的，并且能够抵抗渗透性企图。

橘皮书（美国国防部，1985）是第一本评估计算机安全产品（操作系统中）的指南，在计算机安全的发展中有着巨大的影响力。在橘皮书中，功能性和承诺被捆绑成预定义的类别，用户只能从这个固定的类别集中选择。但是橘皮书相当死板，并且在解决计算机网络和数据库管理系统的安全评估方面还不是非常成功。因此迫切要求有一套更灵活的标准集。

ITSEC 适应了这种需要，它将功能性和承诺区分开，以便规范特定的评估目标（Target of Evaluation, TOE）。不懂安全的用户现在只需弄清楚特定的 TOE，并比较根据不同 TOE 评估的产品即可。

计算机安全的这个两难处境会以不同的形式出现，解决它是目前计算机安全中最为紧迫的挑战。毋庸置疑，解决的方式不会简单。

同这个基本的两难困境形成对照，安全与易用之间的冲突则采用了工程中简单明了的折中方法。安全对性能的影响是多方面的。

- 安全机制需要额外的计算机资源，这个代价容易被量化。
- 安全会干扰用户熟悉的工作模式，繁琐或不合适的安全限制会导致生产力的浪费。
- 必须花费精力去管理安全，因此安全系统的购买者经常选择具有最好的图形用户界面的产品。

2.3 数据与信息

计算机安全是关于对信息和资源的访问控制。然而，控制对信息的访问有时是很难的，因此常常用更加直接了当的目标，即控制对数据的访问来代替。数据和信息之间的区别很微妙，但它

也是计算机安全中某些更加困难的问题的根源。

数据代表了信息，信息是数据的(主观)解释。

数据依照约定所选择的用来表现我们的概念和真实世界中某些方面的物理现象。我们赋予数据的含义为信息。数据用来传输和存储信息，以及依照形式规则处理数据之后获取新的信息(Brinch Hansen, 1973)。

当信息和相应原数据之间存在紧密联系时，这两种概念可能会产生非常相似的结果。但事情并不总是这样的。将信息通过一个隐蔽信道(covert channel, 参见 8.2.5 节)传输是可能的。在这里，数据是对访问请求的“是”或“否”的回答，而收到的信息却是一个敏感文件的内容。另一个例子是统计数据库的推断(inference)问题(参见 17.4 节)。我们扼要地看一下这个问题，考虑一下国内税务局的税收数据库。这个数据库不仅被税务稽查员使用(他们有权访问个人记录)，也被财政部官员使用以便进行总体规划。财政部官员必须能够访问税收的统计概要，但是没有权利访问个人记录。假定数据库管理系统仅允许对足够大的数据集进行统计查询以保护个人数据记录，但是仍然可能对两个仅相差一条记录的足够大的数据集进行统计查询，以将查询结果结合起来。这样，即使没有对数据进行直接访问，仍然可以获得关于某个人记录的信息。

2.4 计算机安全原则

你可能会听到这样一些声明，宣称计算机安全是一个非常复杂的问题，“像火箭科学”。你千万不要被这样的看法吓倒。如果你有机会用系统化的方法去实现一个计算机系统的安全特性，那么一套行之有效的软件(系统)开发方法以及对一些基本的安全原则的良好理解，将使你会达到事半功倍的效果。然而，如果你事后才想起来要在一个已经非常复杂的系统上再加上安全特性，那么当你被该系统已经采取的各种并未考虑安全要求的设计决策所限制时，你就举步维艰了。遗憾的是，后面的情形经常发生。

现在我们要提出一些基本的计算机安全设计要素，这些设计决定提供了编排本书内容的框架。图 2-2 说明了计算机安全设计空间中的主要维度，横轴代表安全策略的重点(参见 2.4.1 节)，纵轴代表一种保护机制被实现的计算机系统层次(参见 2.4.2 节)。

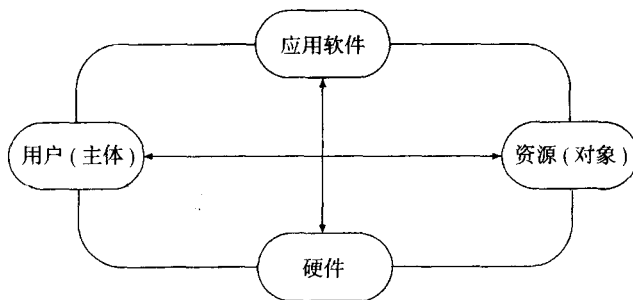


图 2-2 计算机安全的维度

2.4.1 控制重点

现在将重新描述 2.1.4 节中给出的完整性定义。可以说，完整性必须符合一组既定的规则。规则涉及如下几个方面。

- 数据项的格式和内容。比如，一条规则可以规定账目数据库中的余额字段必须包含一个整数。这个规则定义了内部一致性，并且不依赖于访问数据项的用户或者作用在数据项上的操作。

- 可能作用在一个数据项上的操作。比如，一条规则可以规定只有“开户”、“查余额”、“取款”和“存款”操作可以访问账目数据库中的余额字段，并且只有银行工作人员允许执行“开户”操作。这样的规则可能依赖于用户和数据项。
- 允许访问一个数据项的用户。比如，一条规则规定只有账户的持有者和银行工作人员可以访问账目数据库中的余额字段。

我们已经进行了一项重要的总体评述，得出了第一个设计原则。

第1个设计决策 在一个给定的应用中，一个计算机系统中的保护机制应该集中在数据、操作还是用户上？

在应用安全控制时，这个选择是一个基本的设计决策。操作系统传统上集中在保护数据(资源)方面，而在现代应用于中则更多地与控制用户的行为有关。

2.4.2 人一机标尺

图 2-3 给出了计算机系统的简单分层模型。这个模型只准备用作一般性的指导，计算机系统中不一定完全包括所有模型中的层次，也可能多于模型中给出的五个层次。

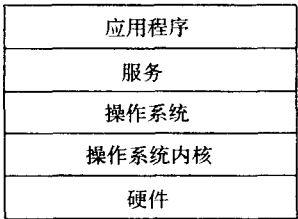


图 2-3 IT 系统中的层次

- 用户运行应用程序(application program)，这些应用程序都满足非常明确的需求。
- 应用程序可以使用像数据库管理系统(DBMS)或对象引用代理(ORB)这样的通用软件包提供的服务(service)。
- 这些服务运行在操作系统(operating system)之上，操作系统执行文件和存储器管理，并控制对打印机和 I/O 设备等资源的访问。
- 操作系统可能有一个协调对处理器和存储器访问的内核(kernel)。
- 硬件(hardware)，即处理器和内存，物理的存储和处理计算机系统中的数据。

安全控制可以被合理地放置于任何一个层次上，我们现在已经解释了第二个基本安全原则的各个方

第2个设计决策 一个安全机制应该被放置在计算机系统的哪一个层次上？

研究已有的安全产品时，可观察到这个模型中从硬件到应用软件每一层上的安全机制。设计者的任务是为每一个机制寻找合适的层次，以及为每一个层次寻找合适的机制。

再来看一下新的设计决策，将一个计算机系统的安全机制想像成一些同心的保护环，其中硬件机制位于中心，应用机制位于外围(图 2-4)。靠近中心的机制趋向于更通用、更面向计算机和更关注对数据的控制访问，外围的机制则更适合解决个别的用户需求。结合前两个安全决策，将参照人一机标尺(man-machine scale)来放置安全机制(图 2-5)。该标尺与数据(面向机器)和信息(面向人之间的差异相关)。

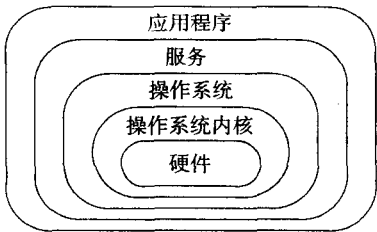


图 2-4 保护机制切面模型

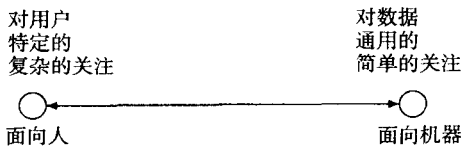


图 2-5 用于访问控制机制的人—机标尺

2.4.3 复杂性与保证性

安全机制在人—机标尺上的位置常常同它的复杂性密切相关。标尺的右侧是简单的通用机制，而应用通常则要求富有特色的安全功能。因此，还必须做出第三个决策。

第3个设计决策 与富有特色的安全环境相比，你是否偏爱简单性和更高的保险性？

这个决策和计算机安全两难困境相关。一个简单的通用机制无法满足特殊的保护需要，但是要在一个功能丰富的安全环境中做出适当的选择，用户必须成为安全专家。因而，安全知识缺乏的用户显然被置于一个不利的位置。

为了获得高保证性，安全系统必须经过尽可能彻底的详细检查，因此在复杂性和保证性之间必然有一个折衷，你想要的保证性级别越高，你的系统就应该越简单。因此，你将会立即注意到以下原则：

富有特色的安全系统和高保证性是很难相互匹配的。

不难理解，高保证性要求遵从系统化的设计实践。事实上，计算机安全是较早以形式化方法为工具来寻求最高保证性的领域之一。

2.4.4 集中控制或分布控制

在一个安全策略内，应该实施相同的控制。如果有一个唯一的中央实体负责安全，那么很容易获得一致性，但是这个中央实体可能成为性能瓶颈。反过来，一个分布式的解决方案可能更有效，但我们必须格外小心，以保证不同的成员实施一致的策略。

第4个设计决策 定义和实施安全的任务是应该交给一个中央实体，还是应该托付给系统中的各个成员？

这个问题出现在分布式系统安全中是很自然的，你可以查看这两种方法的实例。但是，正像 Bell-LaPadula 模型（参见 8.2 节）中强制的和自主的安全策略所论证的那样，这个问题放在大型机系统的环境下研究也很有意义。

2.5 下层

到此，我们已经很简要地谈论了保证性，其中主要探讨了能表达最恰当的安全策略的选择。现在该来考虑一下试图绕过保护机制的攻击者了。每个保护机制均定义了一个安全周界（边界），那些失效后也不会损害保护机制的系统部分位于周界的外部，那些可使保护机制失效的系统部分位于周界的里面。该论断可导致对在 2.4.2 节提出的第 2 个设计决策的直接和重要的扩展。

第5个设计决策 如何防止攻击者访问位于保护机制下面的层？

可以访问“下层”的攻击者处于能够摧毁保护机制的有利位置。比如，如果在操作系统中获得了系统特权，通常就能够改变那些包含有服务层和应用层中安全机制控制数据的程序或文件。如果对物理存储设备有直接的访问权限，就可以绕过操作系统的逻辑访问控制而直接操作原始数据。下面将会给出 6 个例子来进一步说明这一

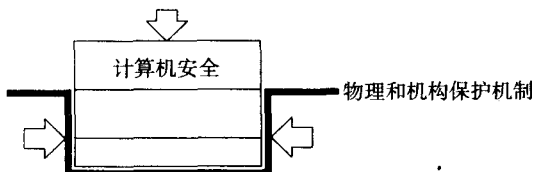


图 2-6 用于控制下层访问的物理的、机构的安全措施

点。安全机制有易于攻击的软肋,并易于受到来自下层的攻击,这些事实表明必须关注这些问题,但没有必要为此而悲观失望。即使在到达一个无法运用计算机安全机制、或不想使用计算机安全机制的阶段时,你仍然能够采取物理的或有组织的安全措施(图2-6)。

1. 恢复工具

如果存储器的逻辑组织由于某物理的存储故障而损坏,即使这些文件的物理表示仍然是完整的,也无法访问文件。恢复工具,像诺顿实用程序(Norton Utility)可通过直接读(物理)存储器来帮助恢复数据,进而恢复文件结构。这样的工具当然也可以用来绕过逻辑访问控制,因为它不关心逻辑内存结构。

2. Unix 设备

Unix 将 I/O 设备和物理存储设备作为文件来管理,因此同样的访问控制机制也可用于这些设备,就像用于文件一样。如果访问许可定义得不好,比如将读访问权授予一个包含读保护文件的磁盘,那么攻击者便可以读取磁盘内容,然后重建文件。读者可以在第6章“Unix 安全”中找到更多的信息。

3. 对象重用(释放存储器)

在一个单处理器多道程序设计系统中,几个进程可以被同时执行,但是任何时候,只有一个进程能够“占有”处理器。任何时候,当操作系统挂起一个运行着的进程去激活下一个进程时,必须执行一个上下文切换(context switch),将被挂起的进程以后要继续执行时所需的全部信息保存下来,并创建与新进程相关的信息。出于安全方面的考虑,必须避免存储残留(storage residue),即在分配给新进程的存储区域中遗留了数据,这将破坏操作系统提供的进程间的逻辑独立性。这可以通过一个固定模式在所有已被释放的存储位置重写或者只允许新进程对它已经写入的存储位置进行读访问来解决。

4. 缓冲区溢出

在一个缓冲区溢出的攻击中,赋予一个变量的值相对于分配给这个值的存储缓冲而言太大,以至于分配给其他变量的存储位置被重写。这个用于修改本来不能逻辑访问的变量的方法将在14.4.1节中进一步解释。

5. 备份

尽责的系统管理员会定期地进行备份。谁能够得到备份磁带,谁就能够访问磁带上的所有数据。逻辑访问控制在此毫无帮助,因此,备份磁带必须被安全地锁起来以保护数据。

6. 内核转储

当系统崩溃时,它会创建一个其内部状态的内核转储(core dump),这样可便于确定崩溃的原因。如果内部状态包含敏感信息,如密钥,并且如果内核转储存放在一个任何人都可以访问的文件中,那么攻击者便可以故意使一个多用户系统崩溃,并在属于其他用户的数据的内核转储中窥探。

2.6 深层阅读

市面上有很多关于计算机安全的书籍。*Russel and Gangemi Sr. (1991)*提供了对该学科的一个非常通俗易懂的介绍。*Amoroso (1994)*介绍了计算机安全的理论基础。*Gasser (1988)*详细讨论了设计安全操作系统的技术细节,综述了 Gasser 在 20 世纪 80 年代建立安全的操作系统的经验(在网上可以找到)。*Pfleeger and Lawrence Pfleeger (2003)*对信息安全进行了综述性的介绍,并为进一步阅读提供了许多有价值的指导。

介绍专用操作系统安全特性的书一般比较昂贵,而且大多关注与管理这类系统相关的问题,

如要调用的窗口及可选择的选项等，对安全实现的方法没有太多深入的介绍。*Park(1995)*关于AS/400的书是一个例外，这本书深入讨论了操作系统的技术细节，并且说明了较低层的代码怎样损害操作系统提供的安全。此外，*Brown(2000)*也出版过关于Windows安全的书。关于Unix安全较好的书有*Curry(1992)*，*Ferbrache and Shearer(1992)*以及*Garfinkel, Spafford and Schwartz(2003)*的著作。更专门的话题，可以在相关章节的后面找到更多资源。

为了正确认识一个研究领域的现状，还应该了解它的历史。*Ware(1979)*和*Anderson(1972)*曾发表过两篇很有影响力的报告，它们引发了计算机安全领域中的许多研究。如果无法查阅这两篇报告，还可以参看*MacKenzie and Pottinger(1997)*提供的有关计算机安全早期历史的总结。

2.7 练习

练习 2.1 收集安全概念定义。可以从美国 TCSEC 计划和英国 ITSEC 计划开始：

<http://www.radiunm.ncsc.mil/tpep/process/fag.html>

<http://www.itsec.gov.uk>

许多大的 IT 公司在自己的网站上也有关于安全的网页。

练习 2.2 解释不可连接性和匿名之间的关系。

练习 2.3 写一篇短文讨论数据与信息之间的区别，并用你自己的例子证明控制对数据的访问并不一定意味着就是控制对信息的访问。

练习 2.4 医疗记录构成了特殊的安全问题。假设可以在线访问你的医疗记录。一方面，这是敏感信息，并应受到保护，防止泄露。另一方面，在紧急情况下又希望任何人均可以进入你的记录。你会如何使用预防、检测和恢复以确保你的记录安全？

练习 2.5 为维护存放在一个计算机系统中的考试结果，草拟一个安全策略。你的策略至少应考虑学生、讲师和行政人员的进入需求。

练习 2.6 为你使用的电脑系统确定可以纳入潜在安全机制的软件组件。

练习 2.7 试讨论：是一个良好的图形用户界面还是一个适当的标准，对购买安全产品更为必需？

练习 2.8 试寻找更多实例：已经进入下层的攻击者可以直接绕过本层的安全机制。

练习 2.9 在分析个人计算机的安全性时可能会用到安全周界的识别。在你的分析中，试考虑什么时候合适假定存放 PC 机的房间、PC 机本身或者 PC 机内的一些安全模块都位于安全周界的范围内。

第3章 身份识别与认证

安全系统有时必须检查请求服务的用户的身份。认证就是验证用户身份的过程。出于以下两个原因需要验证用户身份：

- 用户身份是访问控制决策中的一个参数。
- 在把与安全相关的事件记录到审计日志时，需要记录用户身份。

第15章将解释，为什么基于用户身份的访问控制并不总是必需的或者是合适的。然而，在审计日志中使用身份却有非常充分的理由。本章主要讲解用户身份识别与认证，因为它是当前操作系统的标准。分布式操作系统的认证将在第12章中讨论。

目标

- 回顾一种极为熟悉的认证机制，以此来了解一些一般性的经验。
- 获得口令保护的基本常识。
- 懂得安全机制有赖于管理措施才能有效实施。
- 理解在计算机安全中应用抽象所带来的危险。

3.1 用户名与口令

从字面意义上来讲，你第一次接触计算机安全是从你登录计算机并且被要求输入你的用户名(username)和口令(password)的时候开始的。第一步称为身份识别(identification)，即声明你是谁。第二步叫做认证(authentication)，即证明你就是所声称的那个人。为了将“认证”这个词的用法同其他解释区分开来，本书特别指：

实体认证 验证一个被声称的身份的过程。

一旦你输入了用户名和口令，计算机就会将你的输入同存储在口令文件中的对应条目进行比较。如果你输入了有效的用户名和对应的口令，则登录成功。如果用户名或口令不正确，则登录失败。登录失败之后，通常会再次显示登录界面，这样你可以再次尝试。有些系统会保存登录失败次数，一旦登录失败次数到达了某个阈值，系统就会阻止或延迟当前用户登录。为了减小攻击者使用已有用户登录无人看管的计算机的可能性，可以不仅在会话开始时要求认证，也可以在会话期间定期要求认证(重复认证)。你也可以选择锁住(lock)屏幕，或者在计算机长时间处于空闲状态时自动关闭会话。

经验教训

反复认证解决了计算机安全中一个为人所熟知的问题，即 TOCTTOU(time of check to time of use)。操作系统在会话开始时就检查用户身份，但是在后期会话过程中一直使用这个身份来进行访问控制决策。

从前，你可能在这样的登录界面上输入过用户名和口令。这个界面有友好的欢迎消息以及你即将登录的系统的相关信息。今天，谨慎的系统管理者不会让太多的消息对外部世界可见，而且把欢迎信息替换成了警告：未经授权者请勿使用。比如说，Windows 提供了显示法律公告(legal notice)对话框的选项。在登录操作进行之前，用户必须确认这些警告信息。

今天，大多数的计算机系统都将基于用户名和口令的身份识别和认证作为它们的第一道防

线。对于大多数用户而言,这种机制已经成为他们在自己的计算机上启动会话过程中一个不可或缺的部分。这样,我们就有了一个被广泛接受而又不难实现的机制。在另一个方面,管理口令却要付出极大的代价,而且得到一个有效的口令是获取对计算机系统未授权访问的常用办法。因此,让我们来分析一下作为认证机制的口令的实际安全性。

我们将要探讨以下问题:

- 遗忘口令。
- 口令猜测。
- 口令欺骗。
- 口令文件泄漏。

不要忘了用户在口令保护中起着重要的作用。当你泄露了口令,比如将口令告诉别人,或者把口令记下来放在别人能够找到的地方,都会危及到认证的安全。为了提醒用户口令受到威胁或者最近受到的攻击,在成功登录之后,系统应能够显示上次成功登录的时间以及自上次成功登录之后登录失败的次数。当然,我们自始至终都假设已经为用户账号设置了口令。如果系统管理员或用户忘记了设置口令,攻击者将会省去找口令的麻烦。

3.2 口令管理

口令是用户和对用户进行认证的系统之间共享的秘密。因此,你怎样引导系统将口令存放到正确的位置,而不是其他某个地方呢?在企业中,可以将用户叫到办公室并收集他们各自的口令。如果这种方法不可行,那么还可以通过信件、电子邮件或者电话将口令告知用户,或者直接由用户通过网页输入口令。现在,你必须考虑谁会在中途截取消息,最重要的是谁会利用这些信息。比如,一封包含在线银行账号的信件会被窃取,或者一个冒名顶替者会通过电话索取另一个用户的口令。当远程用户尚未获得口令的时候,你怎样来认证用户呢?这就需要解决以下问题:

- 不要将口令交给呼叫者,而是回拨在你文件(比如,公司内部电话簿)中经过授权的电话号码。
- 回拨其他人,比如呼叫者的经理或者部门安全长官。
- 发送一次性登录口令,这样用户就必须在登录之后将口令更改为发送者不知道的口令。
- 通过邮差专员发送邮件。
- 通过其他渠道获取确认信息从而激活用户账号,比如通过网页输入口令并通过手机短信发送确认信息。

当创建一个新的用户账号时,你可能要忍受获取口令所带来的延迟。然而,当你正在执行一项任务并且意识到自己忘记了口令时,你需要立即采取补救措施。重新设置口令的过程跟上述过程极为相似。但是,组织机构必须设置一个热线服务台来随时处理用户请求。在全球性的组织机构中,这样的热线服务必须是全天候的。从事这项服务的人员必须接受一定的安全培训。因而,口令支持可能成为主要的开销因素。

经验教训

安全机制或许会导致无法为合法用户提供服务。你的安全方案应该能够有效处理这种状况。

3.3 选择口令

口令的选择是一个至关重要的安全问题。尽管你无法排除攻击者猜测出有效口令的威胁,但是你可以努力使这种事情的发生几率尽可能的低。要想知道如何做到这一点,你必须知道到攻击者基本上遵循的以下两种猜测策略。

- 穷尽搜索(蛮力):在一定长度范围内,尝试有效符号的所有可能组合。
- 智能搜索:在有限的名字空间内搜索,比如,尝试那些与用户有联系的口令,像名字、朋友或亲友的名字、汽车品牌、汽车注册号以及电话号码等。或者,尝试那些比较流行的口令。这种攻击方法的典型就是字典攻击(dictionary attack),利用在线词典猜测口令。

那么,你的防卫措施是什么呢?

- 更改默认口令:当系统被交付使用时,它们通常会提供一个默认账户,比如具有默认口令“manager”的默认账号“system”。这种办法有助于领域工程师安装系统,但是如果不改默认口令的话,攻击者就会很容易地进入系统。在刚才的例子中,攻击者甚至能获取一个具有特权级的账号。
- 口令长度:为了防止穷尽搜索,必须预先设定口令的最短长度。遗憾的是,长久以来 Unix 操作系统最大口令长度只能设置为八个字符。
- 口令格式:在你的口令中尽可能包括大小写字符、数字以及非字符符号。口令空间的最小尺寸为 $|A|^n$, 其中 n 为口令的最小长度, $|A|$ 为用于创建口令的字符集的尺寸。
- 避免显著口令:当你发现攻击者配备有一系列的常用口令或当你发现字典攻击已经极大地扩展了“显著”口令的范围时,你大可不必吃惊。而今,你可以找到几乎每一种语言的在线词典。

系统怎样才能进一步提高口令的安全性呢?

- 口令检查器:作为系统管理员,你可以针对某些“脆弱”口令字典使用口令检测工具来检查口令,并且阻止用户选择这类弱口令。这种方法效仿了字典攻击,并且先于针对系统的字典攻击。
- 口令生成:一些操作系统包含了口令生成器。这些口令生成器能够生成随机的,但易于推断的口令。用户不允许使用自己的口令,而必须采用系统推荐的口令。
- 口令老化:在许多操作系统中,口令的有效日期是可以设置的,这样就迫使用户必须定期修改口令。有些附加机制能够防止用户选择以前使用过的口令,比如,系统保存了最近使用过的 10 个口令。当然,通过足够多次地修改直至以前的口令可以被系统接受,执着的用户仍旧可以使用他们最喜爱的口令。
- 限制登录尝试:系统能够监控不成功的登录尝试并做出反应,完全锁定用户账号,或至少锁定一段时间,以阻止或阻碍进一步的登录尝试。用户账号被锁定的时间可以随着登录失败的次数按比例增长。

依照刚才所说的,似乎只要用户使用由系统产生的长口令、口令中混合使用大小写字母和数字符号,并定期修改口令,即可以达到最高度安全性。这种办法真的有用吗?在实际中,我们获得所期望的安全性了吗?

用户难以记住冗长而且复杂的口令。相反,用户会把这类口令写在纸上,并且放在靠近计算机的地方,而这些地方最便于合法用户和潜在的入侵者。安全管理员必须厉行的公务就是查找粘贴在计算机终端上写有口令的纸条。在频繁更换口令的时候,也要考虑到类似问题。那些觉得严格的口令管理制度难以遵循的用户,可能会选择易于记忆的口令,而这些口令也更容易被猜测到。他们可能很快恢复他们最喜欢的口令,或者对这些口令进行简单的、易于推断的修改。如果你每个月都必须更改口令,把月份添加到你最喜欢的口令后面(01~12 的两位数,从 JAN~DEC 的三个字符,任由你选择),这样你就得到了一个你能够记住的口令了。当然,一旦攻击者发现这些口令中的一个,他就很容易猜测出下一个。

接下来,你必须面对这样的事实:即使是那些对安全非常重视的用户,不使用弱口令、不把

口令写在纸条上，都会不可避免地经常忘记口令。这会干扰用户的工作。为了获取新的口令，用户需要联系系统管理员。这又会干扰系统管理员的工作，从而为新的攻击打开便利之门。迫使系统管理员解除口令是一种尝试和检测系统入侵的方法。成功的系统入侵方法更多的是基于社交方法而非精湛的技术(Mitmic and Simon, 2002)。

经验证明，人们更容易记忆那些他们经常使用的口令。因此，在那些需要频繁输入口令的场合使用口令很有效，但是在那些很少需要输入口令场合却不是如此。当你更改口令时，建议重复输入几次新口令。另外一个好建议则是：在周末或节假日之前不要更改口令。

经验教训

你一定不能孤立地看待安全机制。事实上，在一种安全机制上强调太多可能会削弱系统。如果安全机制不合适，用户就不能很好地完成工作，那么用户会找出绕过安全机制的途径。通过口令，你已经注意到了口令的复杂性和人的记忆能力之间的折衷了。

3.4 欺骗攻击

通过用户名和口令的身份认证和识别提供了单方认证(unilateral authentication)。用户输入口令，计算机验证用户身份。但是用户知道谁接收到了口令吗？到目前为止，这个答案都是不知道。用户无法确定在线路另一端当事人的身份。

这是一个实实在在的问题，它导致了第二类口令威胁。在欺骗攻击(spoofing attack)中，攻击者可能是合法的用户。他在某个终端或者工作站上运行程序，显示伪造登录界面。毫无戒备的用户来到这个终端并试图在这个终端上登录。受害者通过标准的登录菜单求输入用户名和口令，之后，这些用户名和口令被攻击者保存下来。接下来，执行流程可能被交还给用户。或者，在给出伪造的出错信息之后终止登录，欺骗程序终止执行。在控制流程交还给系统之后，系统弹出真正的登录请求。用户再次尝试，登录成功，并可能完全不知道口令已经受到威胁的事实。

怎样才能应对这类欺骗攻击呢？

- 显示失败的登录次数可以暗示用户已经发生了欺骗攻击。如果第一次登录失败了，但在进行第二次登录时，被告知上一次会话以来没有过不成功的登录尝试，那么就应该留意了。
- 可信路径：确保用户是在和操作系统通信，而不是同欺骗程序在通信。比如，Windows 有一个安全警示序列(secure attention sequence)(Ctrl + Alt + Del)，它将会调用 Windows 操作系统的登录屏幕。在会话开始的时候，即便系统登录屏幕已经显示了，你也应该按下这个安全序列。
- 相互认证：如果用户要求更有力地确信与之通信的系统的身份，如在一个分布式系统中，那么可以要求系统向用户认证自己。

口令缓存

除了欺骗攻击之外，入侵者可能会采取其他的办法寻找口令。我们对登录的描述是相当抽象的。口令直接从用户处传入口令检查例程。实际上，口令会被暂时存储到中间存储位置，比如缓冲区、高速缓存或者网页。这些存储位置的管理超出了用户的控制范围，而且口令在这些位置的存放时间可能会比用户预料的更长。

在开发基于 Web 的在线银行服务(Arceneaux, 1996)时，开发者所遇到的一个问题很好地说明了这一点。Web 浏览器会缓存信息，以便用户能够返回到他们最近已经访问过的页面。为了使用在线银行服务，你必须在网页上输入口令。你处理完自己的业务，关闭银行服务应用程序，但是并没

有终止浏览器会话。终端上的下一位用户能够返回到有你口令的页面，并以你的身份登录。作为一种防范措施，建议在完成银行业务后退出浏览器。注意到现在用户被要求参与内存管理行为，这一行为是用户本可以不必涉及的。这是对象重用(object reuse)的又一个实例(2.5节)。

经验教训

抽象是有用的，同时又是危险的。用抽象的术语来讨论口令的安全性很有用。你可以研究有应对关口令格式和口令老化的策略，而不需要知道在你的IT系统中口令是怎样处理的。然而，只在这样的抽象层面讨论口令的安全性问题是危险的，执行方面的缺陷可能会威胁到最安全的策略。

3.5 保护口令文件

为了验证用户身份，系统将用户输入的口令同存储在口令文件(password file)中的值进行比较。对于攻击者而言，这样的口令文件当然是极具吸引力的目标。未加密的口令文件内容泄露或者是文件内容的修改，都构成了第三类口令威胁。即便是经过加密的口令文件泄露也应当关注。那样，字典攻击就能离线进行，从而使像限制不成功登录尝试之类的保护措施失效了。为了保护口令文件，我们可以有以下选择：

- 加密保护。
- 由操作系统强加的访问控制。
- 加密保护和访问控制相结合，或者可能有更强的保护措施来减缓字典攻击的速度。

对于口令保护，我们甚至不需要加密算法，有一个单向函数就可以了。目前，你可以使用以下的初步定义：

单向函数(one-way function) 就是易于计算但难以逆推的函数。也就是说，给定 x 很容易计算出 $f(x)$ ，但是给定 $f(x)$ 却很难计算出 x 。

第11章有关于单向函数的更详细的介绍。单向函数被用于保护被存储的口令已经有相当长的一段时间了(Wilkes, 1968, pp. 91ff)。口令文件中保存的不是口令 x 而是 $f(x)$ 。当用户登录并输入口令(设为 x')的时候，系统调用单向函数 f ，并将 $f(x')$ 同用户的 $f(x)$ 进行比较。如果两个值相同，用户就被成功认证。如果 $f(x)$ 是一个理想的单向函数，从 $f(x)$ 重建口令 x 是不可行的。接下来，我们将使用“加密的”口令这个词，尽管我们实际上只是对口令使用了一个单向函数。

由于离线字典攻击的原因，口令文件不能够是其他人可读的。在字典攻击中，攻击者“加密”了字典中的所有单词，并与口令文件中的各个加密条目进行比较。如果找到匹配的条目，攻击者就知道了那个用户的口令。单向函数可以用来减缓字典攻击的速度。这种考虑支配了 Unix 系统中使用的单向函数 crypt(3) 的选择。该函数使用略加修改的 DES 算法，用全零的数据块作为初始值，用口令作为密钥，重复运行算法 25 次(Morris and Thompson, 1979)。当然，这会给合法用户登录带来一点点性能损失。但是，如果优化了单向函数的速度，同时也会提高字典攻击的性能。

操作系统中的访问控制机制只允许拥有适当权限的用户访问文件和其他资源。只有特权用户才能写口令文件。否则的话，即使口令被加密保护了，攻击者也可以简单地通过修改其他用户的口令来访问这些数据。如果只有特权用户才能读口令文件，那么从理论上讲口令文件可以不用加密保存。如果口令文件中包含了非特权用户也需要的信息，那么口令文件必须包含加密口令。然而，这样的文件仍然可被用于字典攻击。一个典型的例子就是 Unix 的/etc/passwd。因此，许多 Unix 版本将加密的口令保存在一个不可公开访问的文件中。这样的文件被称为影子

口令文件 (shadow password file)。

专用存储格式提供了一种较弱的读保护形式。比如, Windows NT 以专门的二进制格式保存加密的口令。一般用户的攻击企图会被挫败, 但是一个顽固执拗的攻击者将能获得或者推断出必要的信息, 从而找到安全相关数据存储的位置。“通过隐匿而获得的安全性”本身不是非常坚固, 但可以把它加到其他安全机制上, 如口令加密。

但是, 存在这样一种危险, 就是对外国防线的成功突破会摧毁其他所有的部分。1997 年初, 有人惊呼 Windows NT 的口令安全已经被突破了。听上去很严重, 不是吗? 实际情况是发布了一个程序, 能将加密的口令从二进制格式转换成另一种易于阅读的表达形式。激动之余发现根本不是什么大事情。

如果你担心字典攻击, 而又不能隐藏口令文件, 你可以考虑口令加盐 (password salting)。当一个口令需要加密保存时, 加密前将一些附加信息 (盐) 添加到口令中。盐同加密的口令保存在一起。如果两个用户有相同的口令, 它们在加密口令的文件中将有不同的条目。加盐减缓了字典攻击的速度, 因为不可能同时搜索几个用户的口令。

经验教训

你已经看到了三个安全设计的原则:

- 几种机制结合起来可以增强保护。加密和访问控制用来保护口令文件。
- 通过隐匿而获得的安全性只能防御不经意的入侵者。不要对这种策略抱太多的信任。
- 如果可能的话, 将安全相关的数据和那些要被公开访问的数据隔离开来。在 Unix 中, /etc/passwd 包含以上两种数据。影子口令文件能够获得所希望的隔离性。

3.6 一次签到

用口令来区分朋友和敌人的历史已经有几个世纪了。在 IT 环境中, 它们控制对计算机、网络、程序和文件等的访问。但作为用户, 如果为了获取一点信息而在信息空间漫游时必须一遍又一遍地输入口令, 你会觉得很不方便。当坐在计算机前, 需要网络中某台数据库服务器中的信息时, 如果必须做以下一些事情, 你会觉得愉快吗:

- 在工作站上输入第一个口令。
- 输入第二个口令以进入网络。
- 输入第三个口令访问服务器。
- 输入第四个口令访问数据库管理系统。
- 输入第五个口令打开数据库中的一张表。

且不去想可能需要记忆五个不同的口令, 并需要在每一种场合选择一个正确的口令输入, 就是将相同的口令重复输入五次已经够糟糕了。

一次签到服务 (single sign-on service) 解决了这个问题。只要输入一次口令, 系统保存这个口令, 并在需要重新认证的时候取出口令替你完成这件事。这样的一次签到服务为你提供了方便, 但它也引发了新的安全问题。如何保护存储的口令? 前面提到的一些技术不再有用, 因为系统现在需要你的明文形式的口令。

经验教训

系统设计者必须权衡方便性和安全性。易于使用是使得 IT 系统真正有用的一个重要因素。遗憾的是, 许多方便的举措也引入了新的弱点。后面还会遇到因方便性引发的灾难伤害你此类的问题。

3.7 可供选择的方法

如果你不满足口令提供的安全级别,你还可以做什么呢?从一般的观点来看,以下选择都是可取的。作为用户,你可以基于以下信息进行认证:

- 你知道的事情。
- 你拥有的东西。
- 你是谁。
- 你做什么。
- 你在哪里。

1. 你知道的事情

用户必须知道一些信息才能被认证。你已经看到了这种认证模式的第一个例子,口令就是你你知道的事情。另一个例子就是同银行卡一起使用的个人识别码(PIN)或者类似的令牌。第三个例子,考虑你打电话查询银行账户的情形,接听电话的银行职员在告诉你任何信息之前可能会向你询问更多的个人信息,像家庭住址、出生日期、配偶名字等。

在这种认证模式中,获知你秘密的“就是你”。另一方面,如果你把秘密告诉其他人,不会留下任何痕迹。当你所在的单位发生了滥用计算机的事件,有人用你的用户名和口令登录了。你能证明你是无辜的吗?你能证明你没有泄露口令吗?

2. 你拥有的东西

用户必须出示一个物理标记才能被认证。能打开锁的钥匙就是你拥有的东西。这种标记的另一个例子就是用于控制进入公司大门的卡片或身份标签。由于口令管理代价的驱使,大型单位已经引入了用于用户认证的智能卡(smart card)。

物理标记可能会遗失或被盗。如上所述,任何拥有标记的人将具有与合法拥有者相同的权利。为了增加安全性,物理标记通常与你你知道的事情结合起来使用,比如银行卡与PIN一起使用,或者它们包含能够识别合法用户的信息,比如银行卡上的照片。然而,就是这些机制结合起来也不能防止骗子获得必要的信息去假冒合法用户,也不能阻止用户自愿传递这些信息。

3. 你是谁

生物测定技术或许能为个人认证提供最终解决方案。生物测定技术是使用人的独特的生理特性,比如人脸、指纹、虹膜图案(Daugman, 1993)、手臂几何学,或许将来还会采用DNA,来进行认证。在编写该书的时候,人们仍在致力于生物测定技术的开发。

我们将以指纹为例来描述生物认证的原理。首先,需要采集用户的指纹(所谓的“参考模板”)。为了更准确,需要收集好几个模板,可能要收集不同手指的指纹。这些模板被存储在安全数据库中。这一过程叫做注册(enrollment)。在用户登录之时,再次读取用户指纹,并将该指纹同存储的模板进行比较即可。采用生物测定技术有以下两个目的。

- 身份识别: $1:n$ 比较,试图从数据库中的 n 个人中找出用户。
- 验证: $1:1$ 比较,对一个给定用户,检查是否存在匹配。

对于每一次认证尝试,基于口令的认证都能够明确地拒绝或接受用户认证。相反,在生物测定中,存储的参考模板几乎从来都不能精确匹配实际测量的模板。我们需要一个计算参考模板和当前模板之间相似值的匹配算法(matching algorithm)。如果相似值超过了预先定义的阈值,就接受用户。因此,我们必须面对一个新的问题,假阳性(false positives)和假阴性(false negatives)。接受错误的用户(假阳性)是一个明显的安全问题。拒绝合法用户(假阴性)则会造成尴尬以及潜在的可用性问题。

通过设置匹配算法的阈值,我们可以交换错误接受率(false acceptance rate, FAR)和错误拒绝率(false rejection rate, FRR),反之亦然(图3-1)。生物测定认证系统的设计者必须在这两种错误之间找到合理的平衡,而这种平衡极其依赖于具体应用。如果FAR与FRR相等,那么阈值为同等错误率(equal error rate)。目前,最先进的指纹识别系统的ERR为1%~2%。虹膜模式识别性能更加优越。总的来说,工业界正尝试在大范围内部署生物测定方案,而在受控环境下得到的结果是否表明了其具有良好的实际应用性能还有待观察。

实际上,技术问题从注册是就开始产生了。注册失败率(failure to enroll rate, FER)指系统无法注册用户的频率,比如指纹皮肤磨损极为厉害以至于无法获取高质量的模板。还有一个问题就是伪造的指纹。指纹以及一般意义上的生理特征或许是唯一的,但决不是秘密。许多地方都可能留有你的指纹,而且经证明要想制作一个可以击溃绝大多数商用指纹识别系统的橡胶手指并非难事(Van der Putte and Keuning, 2000; Matsumoto, 2002)。如果在安全人员在场的情况下进行生物测定认证,那么这只是一个小程序。然而,在认证远程用户时,必须考虑得更充分以便阻止这类欺骗。

最后一个问题:用户会接受这样一个机制吗?在提取指纹时,他们可能感觉自己被当成了罪犯。他们可能也不喜欢用一束激光扫描视网膜的做法。

4. 你做什么

人们往往以一种重复的方式以及个人特有的方式去做一些机械性的工作。在银行业务中,手写签名一直被用于确认用户身份,比如在签署支票和信用卡清单的时候。当然,伪造签名相对更容易。为了获得更高的安全性,用户可以在一个特殊的垫子上签名,垫子可以测量出类似于书写速度和书写力度之类的属性。在键盘上,键入速度以及敲击不同键之间的时间间隔都可以用于认证单个用户。如前所述,认证系统必须建立起来,使得假阳性和假阴性减少到相关应用可以接受的程度。

5. 你在哪里

当你登录时,系统可能还要考虑你在哪里。一些操作系统已经这样做了,并且仅当你是从某个特定的终端登录时才给予访问权限。比如,系统管理员只能从操作员控制台登录,而不能从任意的用户终端登录。类似地,作为用户,你只能从你办公室的工作站上登录。在移动通信和分布式计算中,这种决策使用得更为频繁。如果在认证过程中需要确定精确的地理位置,那就需要使用全球定位系统(Global Positioning System, GPS)。在处理用户的登录请求时确定用户的位置,也有助于解决以后关于用户真实身份的争端。

6. 经验教训

口令并不认证人,成功的认证仅仅表明了用户知道一个特定的秘密。没有办法区分合法用户和已经获取到该用户口令的入侵者。

3.8 深层阅读

Morris and Thompson(1979)介绍了Unix口令的安全历史,在文章中还可以找到非常有趣的

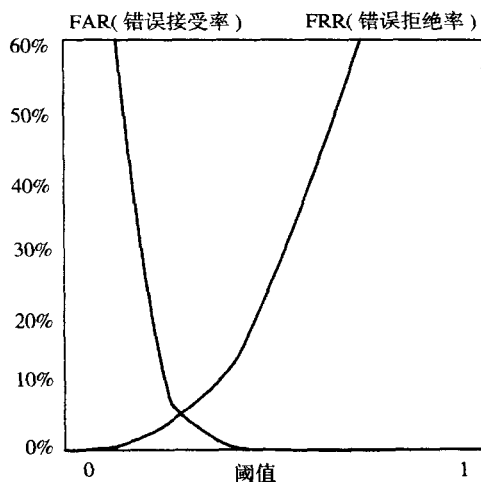


图 3-1 作为匹配函数阈值的典型的 FAR 和 FRR 值

关于口令选择的统计。*Feldmeier and Karn(1990)*介绍了 Unix 口令安全后来的发展。事实上,几乎所有关于计算机安全的书都包含有大量的有关正确选择口令的建议,并且阐述了口令安全的重要性。在因特网上,你可以找到大量的口令破解器(cracker)。分析这样的程序就能帮助你破解程序搜索的口令类型、所使用的字典的大小和复杂性有一个清楚的认识。未经明确允许而运行这样一个程序可能与你单位的规章制度以及许多国家的刑法相抵触。

3.9 练习

练习 3.1 检查你计算机系统上的口令方案。它在口令长度、口令格式以及口令期限方面有规定吗?口令是怎样存储在你的系统中的?

练习 3.2 假定只允许你使用 26 个字母来构造口令:

- 如果口令长度最多为 n 个字符, $n=4, 6, 8$, 不区分大小写,可能有多少个不同的口令?
- 如果口令长度最多为 n 个字符, $n=4, 6, 8$, 区分大小写,可能有多少个不同的口令?

练习 3.3 假定口令长度为 6 个字符,可以使用字母和数字,区分大小写。在以下条件下,蛮力攻击平均所需要的时间分别为多少?

- 检查一个口令需要 1/10 秒?
- 检查一个口令需要 1 微秒?

练习 3.4 假定你只允许使用 26 个字符来构造长为 n 的口令。进一步假设你在区分大小写和不区分大小写的两个系统中使用了此口令,试给出猜测出大小写区分的口令所需要的最大尝试次数。

练习 3.5 试编写一个口令生成程序,以一个长度为 s 的随机数为种子作为输入。在下面的实验中,取 $s=8, 16, 32, 64$:

- 让不同的用户用你的方案生成口令,并监控不同用户选择了相同口令的事件。
- 生成一个口令并对其加密。通过尝试随机种子的所有值试图找出原来的口令。在找到口令之前,你需要的猜测次数的期望值是多少?

练习 3.6 口令由用户输入并由计算机检查。因此,在用户和计算机之间一定有通信的通道。到目前为止,我们一直非常抽象地看待这个通道,假设它们存在并且足够安全。什么时候这种假设是合理的?什么时候是不合理的?

练习 3.7 如果你每次都要求使用几个口令,你可以考虑将它们保存在一个口令簿(password book)里。口令簿是一个含有口令的受保护文件,对口令簿的访问可以通过一个主口令(master password)来控制。这样一种方案能提供任何实际的好处吗?

练习 3.8 *Hellman(1980)*提出了口令猜测在时间和存储器之间存在一个折中。令 N 为可能的口令数目。在一个使用 N 次试加密的预计算步骤中,构造出一个具有 $N^{2/3}$ 个表项的表。如果你以后需要找出一个给定的加密口令,你需要 $N^{2/3}$ 个测试加密。

当长度为 6 的口令选自于 5 比特的字符集时,你需要多少内存空间?如果一次编码耗时 1 毫秒,你多快能找到口令?

练习 3.9 对商用生物测定认证系统进行调查。那些系统的错误接收率、错误拒绝率以及同等错误率各是多少?

第4章 访问控制

现在你已经登录到系统上，创建了新的文件，并且想保护你的文件。你的文件有些可能是公用的，有些文件打算给有限的用户使用，而有些可能是私有的。你需要一种表达你想要的访问控制策略的术语和一种执行访问控制的机制。本章将介绍一些访问控制的术语，在第8、9两章将详细研究具体的访问控制策略。

目标

- 介绍访问控制的基本模型。
- 考虑几组访问操作，重视用直觉来代替术语的实际定义的危险性。
- 给出独立于特定安全策略的基本访问控制结构。
- 定义偏序(partial ordering)和格(lattice)，这是常用来表达安全策略的数学概念。

4.1 背景

在深入访问控制的细节之前，先来细想一下计算机系统以及计算机系统的使用在过去几十年中的发展方式。计算机系统处理数据，并且协调对存储器、打印机等共享资源的访问，它们必须提供对数据和资源的访问控制，尽管主要是为了完整性而不是机密性。传统的多用户操作系统为众多不同的用户提供普通服务。就其本质来说，这些操作系统具有简单和通用的访问操作，并不牵涉及他们所处理的文件的意义。现代桌面操作系统支持单个用户完成他们的工作，在这种情况下，你会发现十分复杂的极其依赖于特定应用的访问操作。用户对他们程序执行时较低层的细节并不感兴趣。毋庸置疑，若将高层的安全需求映射到低层的安全控制上可能十分困难，简而言之，你将会目睹从通用(general purpose)计算机系统到(灵活的)专用(special purpose)计算机系统的转变。在比较本书中介绍的不同的访问控制模型时，需将这个趋势牢牢地记在心里。

4.2 认证和授权

为了讨论访问控制，我们首先必须给出一个合适的术语。“访问”的本质含义是一个活动的实体、一个主体(subject)、一个主角(principal)，使用某种特定的访问操作(access operation)去访问一个被动的客体(对象)，同时有一个引用监控器(reference monitor)(参见第5章)允许或拒绝访问，图4-1给出了访问控制图。

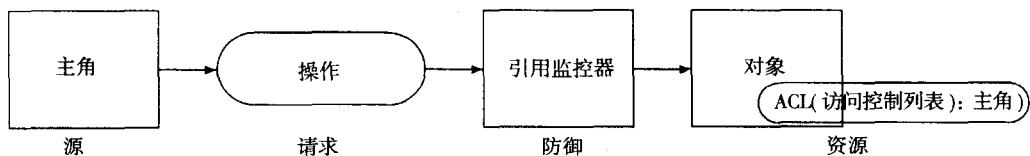


图4-1 访问控制的基本模型

访问控制接着包括有两个步骤，即认证和授权。摘自 *Lampson et al. (1992)*。

如果 s 是一个声明，主角就会在认证过程中回答“谁说的 s ”这个问题。这个主角做出了

声明；这是他们赞同的。同样的，如果 o 是一个客体，主角就会在认证过程中回答“谁是被信任可以来访问 o 的”。

虽然在安全性著作里，对作访问控制的实体有两个术语，即主体和主角，但没有一致的方来区分这两个概念没有区别。一边是主体和主角的关系，另一边是主体/主角和计算机系统用户的关系，这更加混淆了描述。为了能区别这两个概念的意思，我们不妨从有关操作系统安全的早期文献中找到一点提示。

主体代表被我们称之为主角的用户进行操作。在认证过程中，访问是基于主角的名称的，这个名称在认证过程中用一种不可伪造的方式与主体绑定在一起。因为访问控制结构识别主角，所以主角名称必须全局唯一，可读，可存储，能够很容易和可靠地和已知的人联系在一起(Gasser, 1990)。

这个引用阐明了传统的基于身份的访问控制(identity-based access control)，这种访问控制中的安全策略最终和用户相联系。这个仍然是被商用操作系统支持的最普遍的类型，但是，如 15 章讨论那样，它不再是访问控制的唯一范例。在与上面提到的传统相一致的通用框架中，我们在谈到安全策略时使用主角，在谈论执行安全策略的操作系统时使用主体。

主角 主角是一个实体，这个实体能被授予访问对象的权限或做出影响访问控制决策的声明(Gasser et al., 1989)。

主体 主体是 IT 系统的活动实体。

为了访问决策的目的，主体必须和主角绑定在一起。当主体请求对一个访问受保护的對象时，引用监控器会检查与主体绑定的主角是否有权访问对象。因此我们说主体“代表”一个主角。操作系统中的典型的例子是主角就是用户标识。允许访问某个给定对象的主角可以被存储在隶属于某个对象的访问控制列表中(图 4-1)。主体的典型例子就是一个在用户标识(主角)下运行的进程。当然，主角不需要描述用户或用户特性。Java 中的访问控制主要参数是源码(15.3 节)，主体和主角的关系定义如下(Gong, 1999)：

术语主角表示了一个与主体相联系的名字。因为主体可能含有多个名字，一个主体本质上是由一组主角组成。

典型的客体(对象)是文件或资源，像存储器、打印机或计算机网络中的节点。但这并不是要把系统中的每个实体都明确地分为主體或对象。根据不同的情况，实体可能是某个访问请求的主体，而又是另一个访问请求的对象。术语“主体”和“对象”只是区分一个访问请求中的主动方和被动方。主体和对象给出了关注控制的两个选项，你可以规定以下任一种：

- 一个主体允许做什么。
- 可以对一个对象做什么。

这是 2.4.1 节中第 1 个设计决策的例子。通常，操作系统的主要任务是管理文件和资源，即客体(对象)。在这样一种情况下，你遇到的主要是采用第二种方法的访问控制机制。然而，我们刚才已经提到了面向应用的 IT 系统，像数据库管理系统，直接面向终端用户提供服务。这种系统可以适当加入控制主体行为的机制。

在下面的段落中，要涉及：

- 主体集合 S 。
- 对象集合 O 。
- 访问操作集合 A 。

我们不需要对这些集合作更详细的解释了。

4.3 访问操作

从对存储器读写到面向对象系统中的方法调用, 这些访问操作均不相同, 这取决于你如何看待一个计算机系统。类似的系统可以使用不同的访问操作。更糟糕的是, 在看似相同的操作上可能附加不同的含义。本书将从该领域早期的重要文献中研究一些典型的访问操作集。

4.3.1 访问模式

在最基本的级别上, 主体可以观察或者改变对象, 因此, 我们定义了两种访问模式(access mode)。

- 查看(observe): 查看对象的内容。
- 修改(alter): 改变对象的内容。

尽管大多数访问控制策略可以用查看和修改的观点来表达, 但这种策略描述通常离他们正在处理的应用太远了, 使得难以检查正确的策略是否已经执行了。因此, 通常会找到一组更为丰富的访问操作集。

4.3.2 Bell-LaPadula 模型的访问权限

在下一个复杂级别上, 你会发现 8.2 节中讨论的 Bell-LaPadula 安全模型的访问权限(access right)和 Multics 操作系统(Organick, 1972)中的访问属性(access attribute), 这是计算机安全历史上的两个里程碑。

Bell-LaPadula 模型有 4 种访问权限: 执行、读、附加(有时候也称为盲写)和写。图 4-2 给出了这些访问权限与查看和修改这两个基本访问模式之间的关系。

	执行	附加	读	写
查看			X	X
修改		X		X

图 4-2 Bell-LaPadula 模型的访问权限

为了弄清楚这样定义的原因, 可以考虑一下多用户操作系统是如何控制对文件的访问的。用户在允许存取前必须先打开一个文件。通常文件可以被打开用于读或者写访问。在这种方式下, 操作系统可以避免像两个用户同时写同一个文件这种可能的冲突。为了效率的缘故, 写访问通常包括读访问, 比如一个编译文件的用户不应该要求打开文件两次, 一次用于读, 一次用于写。因此, 定义写权限是有意义的, 以便它可以包括查看和修改这两种方式。

几乎没有一个系统真正意义上实现了附加操作。在大多数的应用中, 允许用户不查看对象的内容而去改变对象的操作是不实用的。然而审计日志是附加权限的一个有用的例子。写日志文件的进程不需要读文件, 并且可能根本就不应该读文件。

操作系统可以根本不用打开文件而使用文件, 如程序。因此引入了既不包括查看也不包括修改方式的执行权限。你可能要问计算机如何才能不读程序的指令就执行程序呢? 你当然是对的, 而且 Multics 执行属性确实要求执行和读的权限。但是有些操作, 不用读对象的内容就可以用于执行。假设有一个加密引擎在一个特殊的防篡改寄存器中(图 4-3)存有一个主密钥,

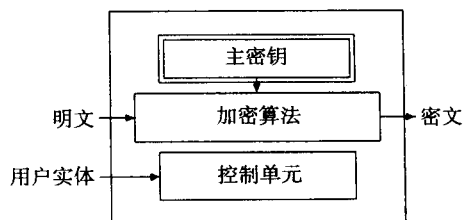


图 4-3 加密引擎

物理上没有办法可以读出这个主密钥，但是访问控制规则可以决定谁能使用这个密钥加密。我们可以不读出这个密钥而调用它，执行权限刚好是解决这样一个问题所需要的。

经验教训

在解释其他人定义的访问操作时，小心使用自己的直觉！

Multics 操作系统区分数据部分的访问属性和目录部分的访问属性。根据对象的类型用不同的方法解释一组给定的访问权限实际上是一种普遍的做法。在命名这些访问属性的时候我们仍然要使用像“读”、“写”和“执行”这样的术语，但是它们与 Bell-LaPadula 模型中的意思不是完全相同的。为了让我们的表示方法保持清晰，我们将用 e, r, a, w 来表示 Bell-LaPadual 的访问权限。图 4-4 给出了访问属性和访问权限之间的对应关系。

数据段		目录段	
读	<u>r</u>	状态	<u>r</u>
执行	<u>e</u> , <u>i</u>	状态和修改	<u>w</u>
读和写	<u>w</u>	附加	<u>a</u>
写	<u>a</u>	搜索	<u>e</u>

图 4-4 Multics 中的访问属性

4.3.3 当前的操作系统

一个较常用的例子是 Unix 操作系统，其访问控制策略用三种操作来表示。

- 读(read)：从一个文件读。
- 写(write)：写进一个文件。
- 执行(execute)：执行一个(程序)文件。

这些操作同 Bell-LaPadual 模型中的那些操作不同，比如，在 Unix 中写访问并不意味着读访问。如果应用于目录时，访问操作采用以下定义。

- 读：列出目录内容。
- 写：创建或重命名目录中的一个文件。
- 执行：搜索目录。

可以看到，Unix 通过控制对目录文件的写访问来控制谁可以创建和删除文件。这些为文件规定的访问权限通过修改文件在目录中的表项来改变。其他操作系统则包含特殊的操作用于删除文件。

我们把 Windows 2000 操作系统作为使用这种设计决策的最后一个例子。*Brown(2000)* 中标准许可(standard permission)列表是：

- 读控制。
- 删除。
- 写自主访问控制列表(DACL)(修改访问控制列表)。
- 写所有者(修改资源的所有者)。
- 同步(用于多线程程序的同步)。

修改访问权限的操作，如，写自主访问控制列表(DACL)，是设置安全策略时你可能想要使用的一种操作。这种修改主体访问权限的操作在主体的权限被第三方修改时一般被称为允许(grant)和取消(revoke)，而在主体改变自己的访问权限时称为断言(assert)和拒绝(deny)。在授权策略(delegation policy)中，当一个主体调用另一个主体且需要建立被调用主体的权限时，即可进行最令人感兴趣的操作。

第7章中给出了更多的关于 Windows 2000 的访问许可内容。

4.4 所有权

在讨论访问控制时，我们还必须规定谁负责制定这些策略。这儿有两个基本的选择：

- 我们可以为每个资源定义一个所有者，让所有者规定谁可以访问。这类策略可以称为是自主的 (discretionary)，因为访问控制由所有者来处理。
- 系统策略规定谁可以访问。显然，这类策略被称为强制的 (mandatory)。

大多数操作系统都支持资源所有权的概念，并且在作访问控制决策时考虑所有权。它们可能包含重定义资源所有权的操作。

上面给出了自主/强制访问控制的直观解释，这仅是为了说明不应该将它们同计算机安全中常用的自主/强制访问控制的定义相混淆。在计算机安全中，这些术语指的是橘皮书 (美国国防部, 1985) 中所规定的特定的安全策略。再次申明，必须小心直觉导致错误。

4.5 访问控制结构

下面，必须申明允许哪些访问操作。当面对两个竞争需求的时候，我们为取得安全策略必须立即选定使用的结构。

- 访问控制结构有助于表达你想要的访问控制策略。
- 你应该能够核实你的策略已被正确取得。

4.5.1 访问控制矩阵

基本上，访问权限可以用访问控制矩阵 (表) 为每个主体和对象的组合单独定义

$$M = (M_{so})_{s \in S, o \in O}, \text{ 当 } M_{so} \subseteq A$$

M_{so} 表项规定了主体 s 可以施加于对象 o 上的访问操作的集合。这种方法追溯到计算机安全的早期 (Lampson, 1974)。访问控制矩阵也被称为访问许可矩阵 (access permission matrices)。Bell-LaPadual 模型使用访问控制矩阵建立橘皮书 (参见 8.2 节) 中的自主访问控制策略模型。图 4-5 给出了一个具有两个用户和三个文件的访问控制矩阵的简单例子。

- bill.doc 可以被 Bill 读和写，但完全不允许 Alice 访问。
- edit.exe 可以被 Bill 和 Alice 执行，但除此以外他们不能访问。
- fun.com 可以被两个用户读和执行；只有 Bill 能够写这个文件。

访问控制矩阵是一个抽象的概念，当主体和对象的数量很大或者主体和对象的集合频繁改变时，不太适合于直接执行。在这些情况下，采用中间控制层是较为可取的。

	bill.doc	edit.exe	fun.com
Alice	-	{ 执行 }	{ 执行, 读 }
Bill	{ 读, 写 }	{ 执行 }	{ 执行, 读, 写 }

图 4-5 访问控制矩阵

4.5.2 能力

你几乎不会直接访问控制矩阵。访问权限可以和主体保存在一起，也可以和对象保存在一起，这之间有一个选择。在第一种情况下，每个主体被赋予一个能力 (capability)，这是一个说明主体访问权限的不可更改的令牌。这个能力对应于访问控制矩阵中的主体行。我们前一个例子作为能力给出的访问权限如下。

Alice 的能力: edit. exe: 执行; fun. com: 执行, 读。

Bill 的能力: bill. doc: 读, 写; edit. exe: 执行; fun. com: 执行, 读, 写

典型地, 能力是同自主访问控制相联系的。当主体创建新的对象时, 它可以通过授予其他主体合适的能力来允许它们访问这个对象。同样当一个主体(进程)调用另一个主体时它可以将它的能力, 或者它的部分能力传递给被调用的主体。

能力绝不是一个新的概念; 但是到目前为止, 能力还没有成为一个广泛使用的安全机制, 这主要与安全管理的复杂性以及操作系统面向管理对象的传统定位有关。

- 对于谁可以被获准访问给定的对象难以得到全面的理解。
- 很难撤销能力, 或者是操作系统必须被授予这个任务, 或者是用户必须清楚他们已经传递的所有能力。当能力中的权限包括向第三方传递该能力时, 这个问题尤其棘手。

然而, 分布式系统的出现重新激起人们对基于能力的访问控制的兴趣, 在分布式环境中安全策略必须处理在计算机网络的节点进行物理的或虚拟的漫游的用户。

当决定使用能力时, 还必须考虑对它们的保护。将能力存放在什么地方? 如果能力仅在一个单独的计算机系统中使用, 那么仅依靠操作系统的完整性保护(参见第 5 章)就可以了。当能力在网络中传播时, 你还需要加密保护(参见第 11 章)。

4.5.3 访问控制列表

访问控制列表(ACL)将对象的访问权限同对象本身存放在一起, 因此, 一个 ACL 对应了访问控制矩阵中的一列, 并声明了谁可以访问一个给定的对象。ACL 是商业操作系统的一个典型特征。我们前一个例子中的访问权限用 ACL 的形式可表示如下。

bill. doc 的 ACL	Bill: 读, 写
edit. exe 的 ACL	Alice: 执行; Bill: 执行
fun. com 的 ACL	Alice: 执行, 读; Bill: 执行, 读, 写。

访问权限的管理依靠单个主体是相当麻烦的, 因此通常将用户置于组中, 并也可以从用户组取得访问权限。Unix 访问控制模型是基于简单的访问控制列表的, 每个访问控制列表均有三个条目用于分派访问权限给用户、组和其他主角(6.5 节)。

对于配合对象访问管理的操作系统来说, ACL 是一个合适的概念。但是, 如果想总的了解个别用户有哪些许可, 比如要取消某个用户的许可, 需要费力地搜索所有的 ACL。

不管你如何实现访问控制矩阵, 在大型的系统中, 管理用这种矩阵表示的安全策略是一项复杂的任务。尤其是, 要确保矩阵中每一项均正确地表示了我们的意愿是相当繁琐并容易出错的。另外, 仅依靠主体和对象的访问控制只支持相当有限的安全策略。更多的信息可以参考主体用来访问对象的程序, 这些信息可以被恰当地包含进访问控制策略中。这完全不是一个新的思想, 可以从以下对 Titan 操作系统的访问控制所作的评论看到这一点。该操作系统是牛津大学于 20 世纪 60 年代早期开发的(Needham, 1992)。

特别要提出的是, 可以一同使用程序标识和用户标识, 甚至用程序标识替代用户标识作为访问控制决策的一个参数, 这是牛津的人们一直以来觉得不可思议而没有采用的特性。

经验教训

不要认为新技术必然引起新的安全问题, “新”问题时常是老问题的化身, 而它们的解决原则都是已知的。

4.6 中间控制

在计算机科学中,有关复杂性的问题都被间接地解决了(David Wheeler)。同样,这也可以用于访问控制。在用户和目标之间我们引入了中间层,以一种更易管理方式来表示这种策略。

4.6.1 组和否定的许可

接下来的讨论将围绕一个简单的例子展开。假如 Alice 和 Bill 是学习某门课程的两个学生。老师想让这些学生访问课程资料。老师不是把所有的学生单独地放进每页课程资料的访问控制列表,而是把所有的学生放入一组,再把这个组放入各自的访问控制列表。

因此,组是作为一种简化访问控制策略定义的方法。具有相同访问权限的用户被集成组,然后再给授予访问对象的许可。一些安全策略要求用户只能是一个组的成员,其他安全策略则允许用户是多个组的成员。

图 4-6 显示了一种理想状况,所有的访问许可都可以间接通过组成员来实现。通常,安全策略有一些特殊的场合。在这些特殊的场合中,直接赋予某些主体访问一个对象的许可,或者否认一个主体从它的组成员关系中获得许可是较为方便的。否定的许可(negative permission)就是访问控制结构中的一个表项,它规定了主体不允许执行的访问操作。图 4-7 中,用户 u_1 不允许访问对象 o_1 ,而用户 u_3 被准许访问对象 o_5 。

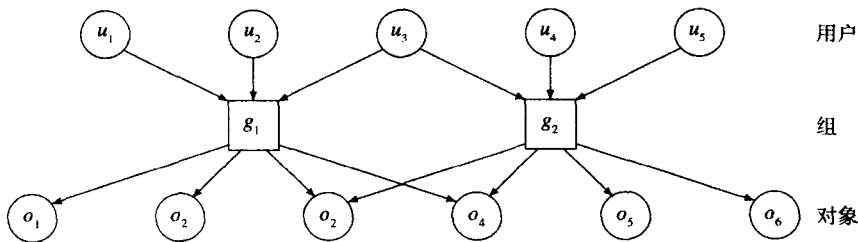


图 4-6 作为一个中间的访问控制列表的组服务

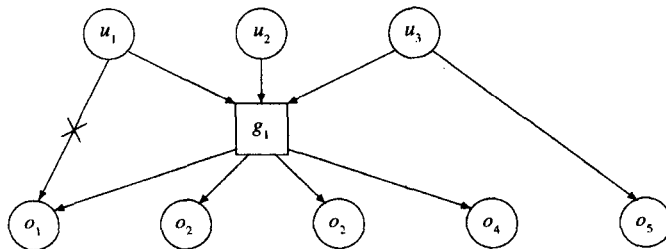


图 4-7 带有否定许可的访问控制

给用户 u_1 的否定许可与给予组 g_1 的肯定许可相矛盾是策略冲突(policy conflict)的一个例子。当详细说明一种策略时,你必须要知道怎样用一个引用监控器解决冲突。如果策略是用访问控制表定义的,一个简单但是广泛使用的算法就是处理列表直到与找到访问要求里给出的主体相符合的第一个条目,依据第一个条目做决策。在列表后面的任何有冲突的表项都将被忽略。

4.6.2 特权

将注意力转到操作上,可以收集权限以执行某些特权操作。典型地,特权和操作系统功能相

联系, 并且涉及像管理系统管理、文件备份、邮件访问或网络访问等行为。你可以将特权视作主体和操作之间的一个中间层(图 4-8)。

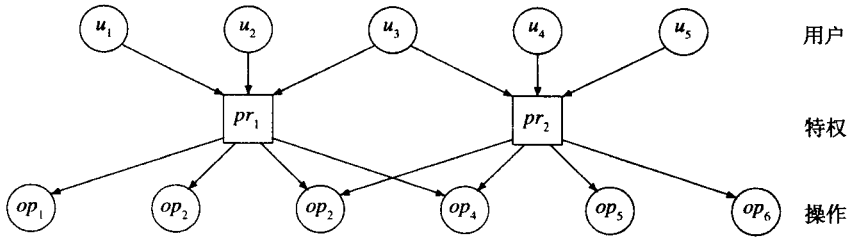


图 4-8 特权作为主体和操作之间的中间层

4.6.3 基于角色的访问控制

特权通常是和操作系统一起预先确定的。一组特定应用的操作(过程)称为角色, 主体从它们履行的角色上获得访问权限。基于角色的访问控制(Role-based access control, RBAC)将主要关注用户和用户执行的任务上。

主体和对象之间的中间层有助于减少访问控制管理的复杂性。中间层可被插入到多个位置上, 这样你可以使用多个层次来构造访问控制。可选的选择的层次包括以下几种。

- 角色: 一个角色就是一组过程, 角色被分配给用户。一个用户可以有多个角色, 多个用户也可以有相同的角色(Sandhu et al., 1996)。
- 过程: 过程是语意比读或写更复杂的高层访问控制方法。过程只能被应用于某些数据类型的对象上。典型实例可参看银行账户之间的资金转移。
- 数据类型: 每个对象均属于一种特定的数据类型, 只能通过为该数据类型定义的过程来访问。通过限制可以访问对象的过程来控制对对象访问是一种常见的程序设计惯例。这是抽象数据类型理论的一个基本概念。

在 4.6.1 节的例子中, 老师可以为学生课程创建一个角色 Student, 并为这个角色指定读课程材料的特权。

尽管这种结构化的访问控制是许多应用非常希望的, 但是许多操作系统还不支持它。突出的例子是 IBM 的 AS/400 操作系统(Park, 1995)和 Windows 2000(参见第 7 章)中的全局组和本地组。RBAC 更常见于数据库管理系统中。

4.6.4 保护环

保护环是用于说明主体和对象之间中间层的一个特别简单的例子。每个主体(进程)和每个对象均依照“重要性”被赋予一个数字, 在一个典型的例子中, 这些数字可能是 0, 1, 2 和 3。进程按照以下规则获得他们的数字:

- 0—操作系统内核
- 1—操作系统
- 2—实用程序
- 3—用户进程

通过比较主体与对象的数字进行访问控制决策。(决策的结果依赖于你使用保护环想要执行的安全策略)。这些数字对应同心保护环(protection ring), 其中环 0 位于中心, 提供最高级别的保护(图 4-9)。如果

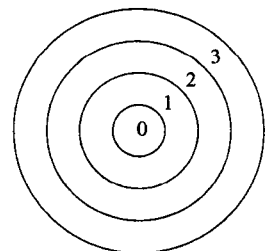


图 4-9 保护环

一个进程赋予了数字 i , 那么我们说进程“运行在环 i ”。

保护环主要被用于完整性保护。保护环已经被用于 Multics 操作系统, 并且开发了特殊的硬件来支持这种安全机制 (Schroeder and Saltzer, 1972), Intel 80x86 系列微处理器在机器语言级提供了类似的特性。Unix 使用了 2 个级别, 根和操作系统运行在环 0, 用户进程运行在环 3。QNX/Neutrino 微内核的系统/用户保护是说明这种保护机制的另一个例子, 它按如下方式给软件组件赋予保护环:

- Neutrino 微内核运行在环 0。
- Neutrino 进程管理运行在环 1。
- 其他所有程序运行在环 3。

包含敏感数据 (如操作系统代码) 的存储位置只能被运行在环 0 或环 1 的进程访问, 5.3.5 节是基于保护环的一个典型策略。我们可以从 Organick (1972, 第 4 章) 和 Pfleeger and Lawrence Pfleeger (2003, 7.2 节) 找到更多的例子。

4.7 偏序

借助保护环, 我们把比较引入了安全策略的评价。保护环是可以决定任意两个环 i 和环 j 谁在最里面的一个简单例子。通常情况下, 这不需要如此。将 4.6.1 节的例子扩展开来, 可以得到以下一个实例。学校的系为一年级学生创建了 Year-1 组来对专用于他们的资源的访问管理, 也为二年级学生创建了 Year-2 组, 为三年级学生创建了 Year-3 组等。一年级学生组包含在所有的学生组中, 但是 Year-1 组和 Year-2 组没有这样的关系。我们力求的最好的就是一个偏序 (partial order)。

定义 集合 (安全级别) L 上的一个偏序 \leq (“小于或等于”) 是 $L \times L$ 上的一个关系, 它是:

- 自反: 对于所有的 $a \in L$, $a \leq a$ 成立。
- 传递: 对于所有的 $a, b, c \in L$, 如果 $a \leq b$ 且 $b \leq c$, 那么 $a \leq c$ 。
- 反对称: 对于所有 $a, b \in L$, 如果 $a \leq b$ 且 $b \leq a$, 那么 $a = b$ 。

如果两个元素 $a, b \in L$ 是不可比的我们记为 $a \not\leq b$ 。

偏序的典型例子如下:

- $(P(X), \subseteq)$, X 的幂集中子集的关系是一个偏序。
- $(N, |)$, 自然数中的除关系偏序。
- 字母表 Σ 上的字符串的前缀关系是一个偏序。如果存在一个字符串 γ 使得 $\alpha = \beta\gamma$, 那么我们就说一个字符串 β 是字符串 α 的前缀。在这种情况下我们记作 $\beta \leq \alpha$ 。

哈斯图是偏序集的图示, 哈斯图是一个有向图, 其节点就是集合的元素。图的边给出了偏序的一个“框架”。也就是说, 对 $a, b \in L$, 我们置一条 a 到 b 的边, 当且仅当:

- $a \leq b$ 且 $a \neq b$, 且
- 不存在 $c \in L$, 因此 $a \leq c \leq b$ 且 $a \neq c$, $b \neq c$ 。

有了这种约定, 当且仅当有一条 a 到 b 的路径时, $a \leq b$ 才成立。图中的边全部连向上面的节点。偏序集合 $((P\{a, b, c\}), \subseteq)$ 的哈斯图在如图 4-10。

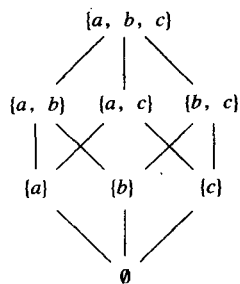


图 4-10 偏序 (格) $((P\{a, b, c\}), \subseteq)$

4.7.1 VSTa 微内核中的能力

VSTa 微内核的能力很好地表现了偏序的作用。因为它们不完全是第 4.5.2 节中定义的能力，所以我们使用 ability 来表示。一个 ability 就是一个正整数中的有限的串。为了表示一个新的整数开始，我们在每个整数的开始前置一个点。因此，一个 ability 就是一个串 $i_1.i_2\cdots i_n$ 。其中 i_1, \dots, i_n 是整数。序列的长度 n 没有限制，事实上 n 可以等于 0，ability 的例子有 .1.2.3.4，或 .10.0.0.5。

通过前缀关系排序的 ability 的顺序构成了一个偏序，在我们当前的例子中，系将 ability .3.1.101 分派给 CS101 课程的学生组。CS101 的课程资料将再次被标记为 ability .3.1.101，一年级的全部资料标记为 .3.1，二年级的全部资料标记为 .3.2，.3 被用作所有资料的所有学生资料的标记。这些 ability 被 $.3 \leq .3.1 \leq .3.1.101$ 联系在一起，但是 $.3.1. \not\leq .3.2$ 。

基于当对象标签是主体标签前缀时的授权访问的策略，CS101 学生将能够访问他们自己的课程资料、一年级的资料和学生的总体资料。

暂且来思考一下以上策略的双重性。如果主体的 ability 是对象 ability 的前缀，就可允许访问。假如这样的话，因为空串 ε 是任何 ability 的前缀，ability“.”即一个后面不带整数的点，就定义了一个可以访问所有对象的超级用户。因此不用给主体赋予 ability，就可以准许它访问所有的对象。

经验教训

访问控制算法比较主体和对象的属性。一定要检查一下如果某些属性缺失会怎么样。故障安全行为要求拒绝访问。但通常情况下不是这样的，你会觉得非常吃惊和不快。

4.7.2 安全级别的格

返回到例子中的最初策略，如果有一个有两组学生可以访问的对象，系就应该采用分给两组的最长的公用的前缀来给文件做标记。例如，如果 Year_1 和 Year_2 能够访问文件，则 ability .3 就可以被用作标签。另一方面，如果有标识着 .3.1 和 .3.2 的两个对象，我们想给能够同时访问两者的主体一个标签，这在当前的系统中还无法实现。

通常，给定标准的机密性策略，仅当主体的安全级别高于对象的安全级别时，这个主体才可以查看这个对象，对以下两个问题我们希望有唯一的答案：

- 给定两个不同安全级别的对象，主体至少具有什么样的安全级别才能允许读这两个对象？
- 给定两个不同安全级别的主体，为使一个对象能够被两个主体查看，该对象能够拥有的最高安全级别是多少？

能让我们回答这两个问题的数学结构已经存在，称为格。格的规范定义如下。

定义 格 (L, \leq) 由集合 L 和偏序 \leq 组成，对于任意两个元素 $a, b \in L$ ，存在一个最小上界 $u \in L$ 和一个最大下界 $l \in L$ ，也就是说：

$$a \leq u, b \leq u, \text{ 并且 } \forall v \in L: (a \leq v \wedge b \leq v) \Rightarrow (u \leq v),$$

$$l \leq a, l \leq b, \text{ 并且 } \forall k \in L: (k \leq a \wedge k \leq b) \Rightarrow (k \leq l)。$$

在安全中，如果 $a \leq b$ ，我们说“ a 受控于 b ”或者“ b 控制 a ”。受控于所有其他级别的安全级别称为系统低 (System Low)，控制所有其他级别的安全级别称为系统高 (System High)。例如，在图 4-10 中的偏序集合 $(P(\{a, b, c\}), \subseteq)$ 是一个格，系统低时集合为空集 \emptyset ，系统高时为集合 $\{a, b, c\}$ 。

无论何时遇见一个安全描述符以某种方式比较的安全系统，如在上述例中的情况下一样，

你会发现如果这些描述符能形成一个格的话，将会很方便的。领会计算机安全的基本内容并不一顶要弄懂格。不过，在阅读了该课题的许多文献后却有助于理解格。

4.7.3 多级安全

保护机密情报的需求推动了 20 世纪七八十年代安全性研究。在这个领域，已经有了管理对保密文件物理访问的规则。机密文件被指定了安全级别 (Security level)，一个用户的许可表明了该用户可以访问哪些文件。强制访问控制 (MAC) 策略和橘皮书的多级安全 (multilevel security) 策略采用了安全级别，并将其应用到 IT 系统中。在早期的大多数版本中，这些策略指的是四个线性排序的安全级别，“非机密的” (unclassified)、“机密的” (confidential)、“秘密的” (secret) 和“绝密的” (top secret 图 4-11)。

绝密的
|
秘密的和
|
机密的
|
非机密的

如果这种线性排序的安全级别，你只能表达一组有限的安全策略。例如，不能限制与秘密工程相关的文件的访问权限，使得只有在 X 中工作的人员才有权访问。任何秘密 (secret) 级别上的人都应有权访问。

图 4-11 在线性顺序中的安全级别

为了能够说明这种控制对特定工程的资源的访问的需要知道 (need-to-know) 策略，我们引入了以下安全级别上的格：

- 令 H 为一个分类的集合，其等级 (线性) 排序为 $\leq H$ 。
- 令 C 为一个种类集合，如工程名字、公司名字、学院的系等。一个间隔就是一个种类集。
- 一个安全标签 (安全级别就是一个二元组 (h, c) ，其中 $h \in H$ 是一个安全级别，而 $c \subseteq C$ 是一个间隔。
- 一个安全标签的偏序 \leq 定义为： $(h_1, c_1) \leq (h_2, c_2)$ ，当且仅当 $h_1 \leq H$ 且 $c_1 \subseteq c_2$ 。

图 4-12 说明了这种结构。有两个等级级别 public 和 private，两个种类 personnel (PER) 和 engineering (ENG)。在产生的格中，以下关系成立：

$$\begin{aligned} &(\text{public}, \{\text{PER}\}) \leq (\text{private}, \{\text{PER}\}), \\ &(\text{public}, \{\text{PER}\}) \leq (\text{public}, \{\text{PER}, \text{ENG}\}), \\ &(\text{public}, \{\text{PER}\}) \leq (\text{private}, \{\text{ENG}\}), \end{aligned}$$

有了这个安全标签的格，你就能实现强制性的需要知道 (最小特权) 策略。为了说明这一点，按照上面提到的简单机密性策略，分析图 4-12 的格。具有安全标签 $(\text{private}, \{\text{ENG}\})$ 的主体将不能够读任何在其标签的间隔中有种类为 PER 的对象。因此，即使标记为 $(\text{public}, \{\text{PER}, \text{ENG}\})$ 的对象也不能读。

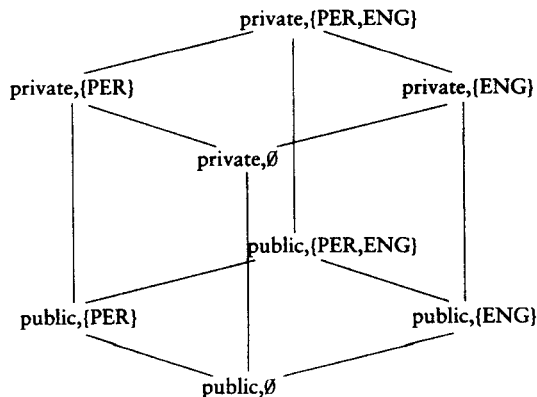


图 4-12 安全标签的格

我们从图 4-11 中简单的等级格来开始安全格的讨论，等级格在政府部门的多级安全策略中是很典型的，并且这样的系统已经被构造出来以非常高的安全度执行这些策略，然后我们加入了间隔来表示种类更多的策略。今天，可看到多级安全系统的应用，因为这种系统安全度高，但它在安全级别中没有等级成分。比如，一种防火墙方法可以使用图 4-13 的格实现内部和外部的严格隔离。

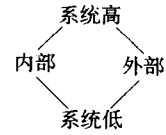


图 4-13 用于防火墙的格

4.8 深层阅读

Denning(1982)、Amoroso(1994)和 Pfleeger and Lawrence Pfleeger(2003)都谈及过基本的访问控制结构和安全格。Lampson(1974)和 Saltzer(1974)的论文是早期较有影响的访问控制(保护)方面的论文。Wilkes(1968)较详细地介绍了 20 世纪 60 年代开发的操作系统的访问控制。更多的关于保护环安全的策略的例子可参看 Nelson(1998)和 Lawrence Pfleeger(2003)的著作。基于角色的访问控制(RBAC)的最新调查发表在 Sandhu(1996)等人的著作中。Sandhu(1993)的著作中可以找到更多有关基于格的访问控制模型的信息，以及如何使用它们来处理机密性和完整性问题的描述。

4.9 练习

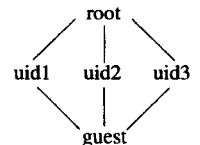
练习 4.1 给定两个比特位来控制在一个目录上的访问操作，怎样才能使拥有 4 种操作？怎样控制文件的创建与删除？怎样用这些操作来实现文件的隐藏？(隐藏文件仅授权主体可见)

练习 4.2 考虑一个有读、写、授权和撤销的系统。用授权命令不仅可以授予其他主体读和写的访问，也可以授权它们访问你所拥有的对象的权利。用哪种数据结构和算法来实现授权和撤销操作，以便你能撤销你所属的某个对象所有的访问？

练习 4.3 讨论：组和角色的不同点是什么，它们是否有本质的区别？

练习 4.4 解释 4.7.1 节定义的 ability 的偏序不能构成一个格的原因。试着把更多的要素加入 ability 组中以使偏序转化为格。

练习 4.5 给定一个当且仅当主体的安全级别控制对象的安全级别时，主体才能访问对象的安全策略。在这种策略下，用下图所示的格可以得到什么结果？



练习 4.6 设 (L, \leq) 是一个安全级别的格，其中 L 是一个有限集合。试说明系统低和系统高的单元系统一定存在于这个格中。

练习 4.7 给安全级别是 public、confidential 和 strictly confidential，以及种类是 ADMIN、LECTURERS 和 STUDENTS 构造一个安全标签的格。在需要知道的策略中，哪些对象对安全标签为 (confidential, {STUDENTS}) 的主体是可见的？ n 个安全级别和 m 个分类可以创建多少个标签？为了方便说明，可假定 $n = 16$ ， $m = 64$ 。

练习 4.8 给定一个用间隔的格作为安全标签的安全策略。当且仅当主体的安全标签是对象安全标签的子集时才能授权访问。假设有 ADMIN, LECTUREERS, STUDENT 这几个类，标签为 {STUDENTS} 的主体能访问以上哪几个对象？为什么标签为 {ADM, STUDENTS} 的主体比标签为 {STUDENTS} 有更多的限制？解释在这个策略中标签为 \emptyset 和 {ADMIN, LECTURERS, STUDENTS} 的角色的意思。

练习 4.9 给定一组类集，实现基于格的需要停止 (need-to-withhold) 策略，该策略可以有选择地从主体收回访问权限。

第5章 引用监控器

上一章介绍的一些基本概念对设置访问控制策略非常有用。现在我们讨论如何实施上述策略。在后续章节中,对策略描述和访问控制系统将有更多的详细介绍。本章给出用以保护操作系统完整性和存储器访问控制的核心机制,重点关注基于分层系统构架底层的访问控制,同时给出设计安全系统的经验教训。

目标

- 给出访问控制的基本概念,如引用监控器(reference monitor)、可信计算基(Trusted Computing Base)。
- 讨论影响设计引用监控器的要素。
- 引入状态和受控调用,它们是两个重要的安全原语。
- 理解在系统底层实现安全机制的原因,对于在系统底层上可以利用的安全机制有一个全面了解。

5.1 引言

计算机安全领域有三个基本概念,它们紧密相关但容易混淆。我们可引用橘皮书(美国国防部,1982)中的词汇表为其定义:

引用监控器 这是一个访问控制概念,指所有主体对对象访问进行仲裁的抽象机器。

安全内核 可信计算基由硬件、固件、软件组成,它们用以实现引用监控器。安全内核必须处理所有的访问控制,必须保护安全内核,使之不被修改,并能够证明是正确的。

可信计算机基(TCB) 计算机系统中全部的保护机制——包括硬件、固件、软件,它们结合起来负责实施安全策略。一个TCB包含一个或多个组件,这些组件在一个产品或系统中实施统一的安全策略。TCB正确实施安全策略的能力完全取决于TCB的内部机制,以及系统管理员对于安全策略相关参数(如用户清除)的正确输入。

因此,引用监控器仅是一个抽象概念,安全内核是它的实现,而TCB在其他的保护机制中包含了安全内核。Anderson(1972)报告论述了引用监控器实现的核心要求。

- 引用验证机制必须能防渗透(tamper proof)。^①
- 引用验证机制必须总能被调用。^②
- 引用验证机制必须尽可能小,能被分析和验证是正确的。

Anderson 研究报告很大程度上曲解了对引用监控器和安全内核的理解。这份报告指出,安全内核包括引用验证机制的实现、对系统自身的访问控制,以及组成管理用户和程序的安全属性组件。图2-3是安全内核在系统底层的实现的有力说明。相反,有时你会发现安全内核代表底层的安全机制。

① 如今,安全学中防篡改(tamper resistant)已经取代了防渗透(tamper proof),以便不会让人产生安全组件牢不可破的误解。

② 这个要求即是为众人所知的完整调解(complete mediation)。

5.1.1 部署引用监控器

原则上, 引用监控器可以放置在分层系统的任何地方; 实际上, 所有可能设计的例子都能在实际应用中找到原型。

- 硬件级: 基于微处理器的访问控制机制将在 5.3 节讨论。
- 操作系统内核级: 超级监视者 (hypervisor) 是一个虚拟机, 它能精确地模拟运行在其上的主机。通过提供独立的虚拟机, 它能用于区分出不同用户与应用程序。Nexus 是基于微软 NGSCB 架构的一个例子 (England et al., 2003)。
- 操作系统级: 第 6 章和第 7 章分别讨论了 UNIX 和 Windows 2000 的访问控制。Multics 上的引用监控器将在 8.3 节讨论。
- 服务层: 包含 Java 虚拟机、.Net 的 CLR (Common Language Runtime, 通用语言运行时)、CORBA 的中间件架构, 以及数据库管理系统 (第 17 章)。
- 应用程序: 对安全需求非常了解的程序员会选择在应用程序中嵌入安全检查代码, 而不是调用底层的安全服务。

我们可以根据应该控制的“应用”来部署引用监控器。系统底层可以提供引用监控器, 如图 5-1a 所示, 这是操作系统典型的访问控制模式。应用程序要求访问受保护资源。引用监控器是操作系统内核一部分, 它要处理所有访问控制。CORBA 中的访问控制与此类似。

同样, 程序在解释器中运行。解释器通过程序处理所有访问控制。解释性语言 Java 就是这种方法的很好的例子, 程序位于引用监控器内部, 如图 5-1b 所示。而解释器处理所有访问控制。

第三种方式便是通过重写程序来嵌入访问控制检查。Erlingsson 和 Scheider (2000 年) 引入的内联引用监控器便是这种例子。如今, 该引用监控器已被放入程序内部 (如图 5-1c 所示)。

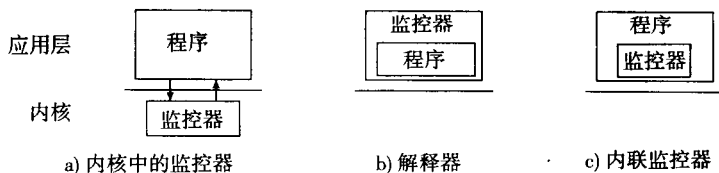


图 5-1 部署引用监控器

5.1.2 执行监控器

引用监控器通过请求的访问信息和被请求访问的对象来决定访问请求。通过在做出访问决策中实际使用的信息, 我们便能区分不同的引用监控器 (Schneider, 2000)。

- 执行监控器 (execution monitor; Schneider, 2000) 通常只浏览执行步骤的历史记录, 并不预测未来执行的结果。该变量在操作系统中通常很常见, 并只保留执行中有限 (少量) 历史信息。
- 当做出仲裁时, 引用监控器能够考虑目标所有的执行可能。静态类型检查 (Static type checking) 便是这种例子。
- 引用监控器通过重写访问对象来确保访问请求符合安全规则。

5.2 操作系统完整性

现在让我们概括地确定一下包含在安全内核中的安全机制。假设操作系统能够实施所有的访问控制策略。只要操作系统如预期的那样运行, 对资源的非授权访问就不可能进行。当然, 这正是对攻击者的暗示。为了避开保护机制, 攻击者可能试图通过修改操作系统使安全控制失效。

现在面临的将不仅是完整性问题——即使当初用户只关心机密性。操作系统不仅是访问请求的裁定者，它本身也是一个访问控制对象。新的安全策略是：

用户一定不能修改操作系统。

这是一个通用安全策略，应当得到大力、有效支持。但更艰巨的任务是，你必须处理以下两个相互冲突的要求：

- 用户能使用(调用)操作系统。
- 用户不得滥用操作系统。

通常用于实现这些目标的两个重要概念是状态信息和受控调用(也称受限特权)。这些概念可被用于计算机系统的任一层次上，不管是应用软件、操作系统还是硬件。但是再次强调，如果攻击者获得较低层次的访问，这些机制可能会失去作用。

5.2.1 操作模式

操作系统保护自己不受用户破坏的第一个前提条件是能够区分代表操作系统的计算和代表用户的计算。状态标志(status flag)可达到这一要求，它允许系统在不同的模式工作。例如，Intel 80x86 有两个状态位，支持四种模式而 UNIX 操作系统能够识别超级用户(根)模式与用户模式。

为什么这些模式有用呢？比如，为阻止用户直接写存储器、破坏逻辑文件结构，操作系统可以仅在处理器工作于超级用户模式时准许对存储位置进行写访问。

5.2.2 受控调用

继续我们的例子，用户想执行一个要求超级用户模式能执行的操作，如写存储器。为处理这个请求，处理器必须转换模式，这种转换应当如何执行呢？如果简单地将状态位改到超级用户模式，会将与该模式相关的所有特权都赋予该用户，但对用户实际做些什么却没有任何控制。因此，理想的做法是系统在超级用户模式只执行一组预先定义的操作，而在将控制交回给用户前返回用户模式。我们称这个过程为受控调用。

5.3 硬件安全特性

在我们的 IT 系统中，硬件是最底层的，它同时是计算机安全与物理安全相联系的地方，自然地，硬件级的安全机制自然成为我们研究的起点。本节将通过 Intel 80386/486 和 Motorola 68000 讨论了微处理器的安全特性。

5.3.1 安全基本原理

将安全性置于两个较低的系统层之一有两个很好的理由(如图 5-2 所示)。如果攻击者从底层进入，任何给定层次上的安全机制都将遭到破坏。因此，为了评估系统安全性，必须证明安全机制不会被绕过。系统越复杂，证明越困难。在系统的核心，有可能找到相当简单的能够经得起全面分析的结构。这就论证了把安全性置于系统核心的第一个原因。

可以在更高的保证级别上评估安全。

微处理器设计几乎就是创建那些对大多数用户来说最有用的操作集的技法。对这些原始操作的正确选择和有效实现决定了总体性能。当实现安全性时，可以遵循相同的办法。决定一般性安全机

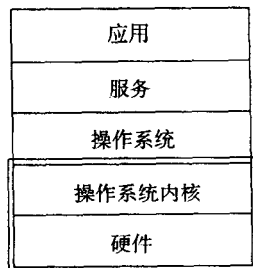


图 5-2 安全内核中的保护

制，并把它们置于系统核心中。这就是把安全性置于系统核心的第二个原因。

把安全性置于系统核心减轻了因安全性引起的性能损失。

把安全性置于系统核心的所有论证都将我们推向人一机标尺的机器端(如图 5-3)。结果是可以预料的。

引用监控器制定的访问控制决策已远离由应用程序制定的访问控制决策。

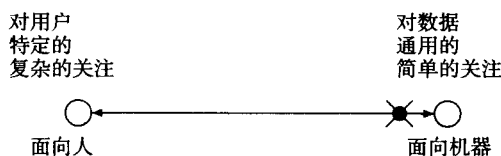


图 5-3 人机标尺中安全内核的放置

5.3.2 计算机体系结构的简单概述

我们假设本书读者对计算机体系结构基本概念已比较熟悉了。如果还不熟悉，可以参考 *Hennessy and Patterson(2002)*^①。就目前而言，用图 5-4 给出的简单计算机示意图描述就可以了。计算机由中央处理单元(CPU)、存储器和连接二者的总线组成。在实际中，这三种实体都有非常精细的结构。

1. 中央处理单元

核心的 CPU 部件包括：

(1) 寄存器(Register)：包含通用寄存器和专用寄存器，

重要的专用寄存器有

- 程序计数器(program counter)：指向包含下一条要执行指令的存储器位置。
- 栈指针(stack pointer)：指向系统堆栈的栈顶。
- 状态寄存器(status register)：允许 CPU 保存重要的状态信息。

(2) 算术逻辑单元(Arithmetic Logic Unit)：执行用机器语言给出的命令；指令执行后可能会在状态寄存器中置位。

系统堆栈(system stack)是存储器中一块特别指定的部分，堆栈可以通过将数据压入栈顶或从栈顶弹出数据来进行访问。为了在不同的进程间切换，CPU 执行上下文切换操作，并在将控制交给新进程前将当前进程的状态，如程序计数器、状态寄存器等保存在堆栈中。

2. 输入/输出

输入设备(如键盘)和输出设备(如显示器)大大增加了计算机的易用性。在输入安全敏感数据如用户名和口令时，需要建立一条从输入/输出设备到 TCB 的可信路径(Trusted Path)。Windows 2000 中的安全警告序列就是一个例子。在用户需要对文档签名的应用中，必须保证显示的文档就是实际需要被签名的文档。

3. 存储结构

下面简要介绍不同的存储结构的安全特性。

(1) 随机存储器(random access memory, RAM)：读/写存储器；必须考虑完整性和机密性问题。

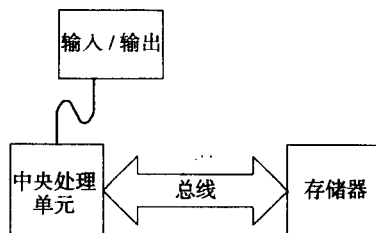


图 5-4 计算机的示意图

① Computer Architecture: A Quantitative Approach(《计算机体系结构：量化研究方法》)，本书由机械工业出版社引进出版。——编者注

(2) 只读存储器(read-only memory, ROM): 有内建的完整性保护机制, 只需要考虑机密性问题; ROM 可能是存储(部分)操作系统的好地方。

(3) 可擦写可编程只读存储器(erasable and programmable read-only memory, EPROM): 可用于存储部分操作系统或密钥; 技术上更复杂的攻击可能会对安全造成威胁。

(4) 一次写入型光盘(write-once memory, WROM): 这种存储结构配备一种允许一次性固化存储内容的机制; 在硬件中可以通过熔断写线上的熔丝来做到。但是还可能遇到“逻辑熔丝”; WROM 可能是存储密钥的一个好地方; 只写磁盘正被用于记录审计踪迹。

进一步, 还可以分为易失性(volatile)存储器和非易失性(nonvolatile), 即永久性(permanent)存储器。前者当断电时会丢失内容, 在物理上这个过程既不是瞬间的也不是完全的。如果断电后立即接上电源, 旧的数据还可能保存在存储器中。即便电源被关闭了一段时间, 旧的存储器内容仍然有可能通过特定的电子技术恢复。为了抵抗这种攻击, 存储器必须用合适的取决于存介质的位模式重复地改写(美国国防部, 1987)。

永久性存储器在断电时仍能保持其内容。如果敏感数据(如密钥)存放在永久存储器中, 并且攻击者能够绕过 CPU 直接访问存储器, 那么必须实现像口令检测保护或物理保护等进一步的措施。比如将光传感器放置在防篡改模块中, 以检测某些操作企图并触发对模块中存储内容的删除。物理保护本身就是一个研究课题, 它已超出本书讨论范围。我们将注意力集中在用户只能通过 CPU 访问内存的情形, 研究 CPU 如何实现机密性和完整性。比如, 如何能够防止计算机病毒用一个受感染的操作系统版本改写一个干净的版本。

有一点很重要, 就是要提醒自己图 5-4 中的存储器是另一个抽象概念。逻辑上, 存储器可能由主存储器、高速缓存存储器和缓冲器组成, 即便是备份介质也可以包括在其中。因为, 一个数据对象可能同时存在于该存储层次结构中的多个位置上。除了在主存储器中有一个永久性的拷贝之外, 还可能有时性拷贝。通常, 这些临时拷贝的位置和生存期不在用户的控制之下。如果这些临时拷贝之一被保存在一个未受保护的存储区域, 对数据对象的安全控制就可能被绕过。

5.3.3 进程和线程

进程是一个正在执行的程序, 因此, 对于操作系统来说进程是一个很重要的控制单元, 当然对于安全来说也是如此。简单地说, 一个进程由以下几个部分组成:

- 可执行代码。
- 数据。
- 执行环境, 如某些相关的 CPU 寄存器的内容。

进程(Process)在它自己的地址空间中工作, 并且只能通过操作系统提供的原语同其他进程通信。进程间的这种逻辑分离对于安全来说是非常有用。但另一方面, 进程间的上下文切换开销很大, 因为操作系统必须将全部的执行环境保存在堆栈中。

线程(Thread)是进程内部的执行流。由于线程间共享一个地址空间, 它们避免了全部的上下文切换开销, 但是也避开了可能的安全机制控制。

5.3.4 受控调用——中断

处理器能够处理由程序错误、用户请求、硬件故障等引起的执行中断, 这种处理机制被称为异常(exception)、中断(interrupt)、陷阱(trap)等不同的名称。不同的术语可能指的是不同类型的事件, 但仍然有相冲突的分类法, 若要进一步了解可参看 *Hennessy and Patterson (2002)*。

接下来, 我们将陷阱用作一般术语, 并且解释怎样将陷阱用于安全目的。陷阱是 CPU 的一

种特殊输入，它包括中断向量表(interrupt vector table)中一个称为中断向量(interrupt vector)的地址。中断向量表给出了一些程序的位置，这些程序用于处理陷阱规定的时间，称为中断处理器，见图 5-5。当一个陷阱发生时，系统在堆栈中保存当前状态，然后执行中断处理器，用这种方式将控制从用户程序那儿接管过来。中断处理器必须保证在将控制返回给用户程序前，将系统恢复到一个正确的状态，比如清除超级管理用户状态位。

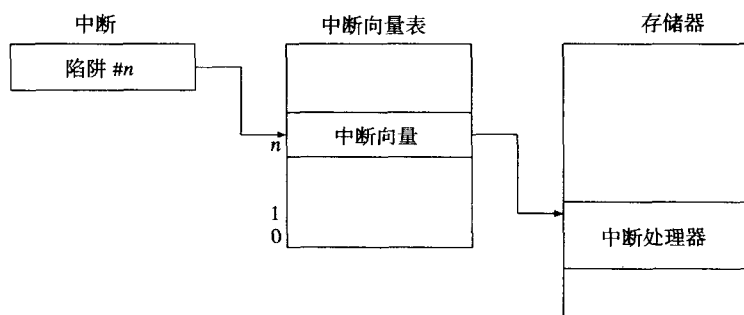


图 5-5 中断处理

在处理器处理当前中断时有可能出现另一个中断。接下来，处理器可能会中断当前的中断处理器。这种情况下的不合适处理可能会导致安全缺陷。一个典型的例子就是：用户可以通过键入 Ctrl + C 中断程序的执行，之后，处理器就可以根据当前进程的状态位一起返回给操作系统。然后用户就可以通过操作系统调用的执行进入超级用户模式。所以，在执行程序前建立中断表以使得中断可以被合适处理是相当重要的。

从以上讨论可以清晰看出，中断表是一个特别引人注目的攻击点，必须受到充分的保护。改变中断表中的入口使它指向一段攻击代码，然后在跳转到正确的中断处理前执行它，这就是病毒创建者使用的策略(图 5-6)。在设计技术中，间接层非常有用，并被频繁使用着。但是，不管什么时候用到这种间接技术，我们都应该看看有没有如上描述的攻击。

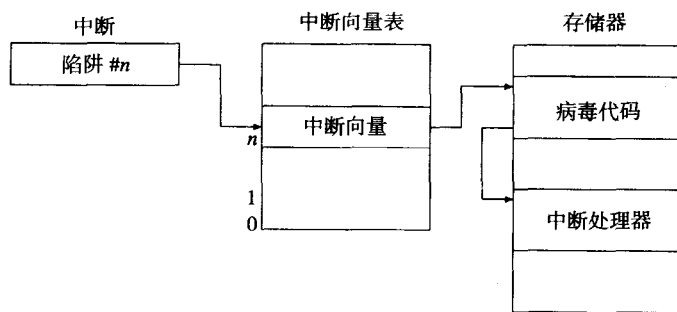


图 5-6 通过中断向量重定向插入病毒代码

经验教训

重定向指针是一种非常有效的攻击模式。

5.3.5 Intel 80386/80486 上的保护

Intel 80386/80486 是 32 位微处理器，80386/80486 上的保护模式支持多任务操作系统的完整性和机密性要求。

Intel 80x86 在状态寄存器有一个 2 比特的域，定义了四种特权级别(保护环，参见 4.6.4 节)，

特权级别只能被运行在级别0上的单指令(POPF)修改。软件可以被指派到如下各个级别上。

- 0—操作系统内核。
- 1—操作系统其他部分。
- 2—I/O 驱动程序等。
- 3—应用软件。

并非所有的操作系统都使用全部的四个级别, 比如 UNIX 只使用 0 和 3 级, Intel 80x86 实现了以下安全策略。

程序只能访问位于自己环内或环外的对象。程序只能在自己的环内调用子例程。

80x86 保存系统对象的信息, 如存储器段、访问控制段或描述符(descriptor)中的门。描述符保存在描述符表中, 通过选择器(selector)进行访问, 对象的特权级别保存在它的描述符的描述符特权级(Descriptor Privilege Level, DPL)域中。选择器是一个 16 比特的域, 包含一个指向描述符表中对象入口的索引以及一个被请求的特权级别(Requested Privilege Level)域(图 5-7), RPL 域的使用参见后文。只有操作系统能够访问选择器。

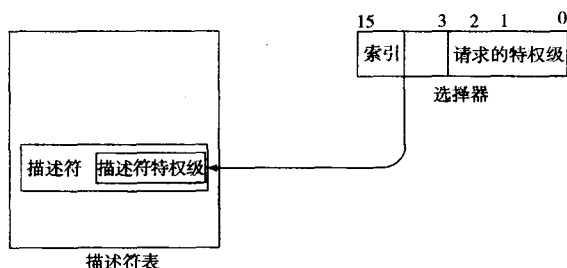


图 5-7 选择器和描述符

包含主体(即进程)信息的系统对象当然也有描述符和选择器。当主体请求访问对象时, 相关的选择器被装入专用的段寄存器。比如, 当前进程的特权级别, 称为当前特权级别(current privilege level, CPL), 就是保存在代码段(CS)寄存器中的选择器特权级别。

毋庸置疑, 我们又一次面对了这样的问题, 就是必须对那些要求较高特权的操作进行访问管理。假设环 3 中的某个应用程序需要环 1 中的一个操作系统例程的服务, 在 80x86 中的这个问题是通过门(gate)来解决的。门是一个指向某个程序(某个代码段中)的系统对象, 在这里, 门具有和它所指向的代码不同的特权级别。门允许对环内中的程序进行只执行访问, 但仍限制向外调用。

为使程序使用门, 门必须和程序位于同一个环内。当通过门调用一个子例程时, 当前特权级别变为门正指向的代码级别。当从子例程返回时, 特权级别恢复到调用程序的特权级别。子例程调用也在堆栈中保存指示调用程序状态的信息或返回地址。为了确定堆栈的合适特权级别, 记住调用程序不能写入内环。但是出于安全的原因, 将堆栈保留在外环也是不能解决问题的, 因为这会使返回地址完全没有保护。因此, 一部分堆栈(多少在门的描述符中规定)要拷贝到一个更有特权的堆栈段中。

困惑的代理人问题

允许外环程序调用内环程序将会产生一个潜在的安全漏洞。外环程序可以请求内环程序将位于内环上的一个对象拷贝到外环上^①。迄今为止, 我们给出的任何安全机制都不能防止这种做

① 如今, 困惑的代理人问题(confused deputy problem)这个术语通常用于描述这样一种情形: 一个没有特权的实体调用一个更高特权的实体做出违反安全策略的行为。

法,事实上它也没有违反我们规定的安全策略。因此,只可能寄希望于扩展原来的安全策略,使其不仅考虑到当前的特权级别,也考虑到调用程序的级别。

在 80x86 中,可以使用选择器中的 RPL 域和调整被请求的特权级别 (adjust requested privilege level, ARPL) 指令支持这种策略。ARPL 指令将所有选择器的 RPL 域改为调用程序的 CPL, 然后系统可以比较 RPL (选择器中) 和 DPL (描述表中), 如果它们不同, 就拒绝完成被请求的操作 (图 5-8)。

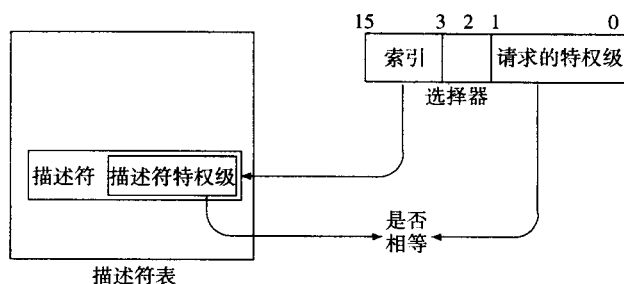


图 5-8 比较 RPL 与 DPL

经验教训

为了在访问控制中达到更高的精确度, 最好在决定访问请求时考虑一下执行历史的某些方面。

5.4 存储器保护

操作系统管理对数据和资源的访问, 通常不涉及对用户数据的解释。多任务操作系统交替执行属于不同用户的进程, 因此操作系统不仅必须保护它自己的完整性, 还必须阻止用户有意或无意地访问其他用户的数据。操作系统本身的完整性通过隔离用户空间和操作系统空间来保护。用户的逻辑隔离可防止用户有意或无意的干扰。隔离发生在两个级别上。

- 文件管理: 处理逻辑存储对象。
- 存储管理: 处理物理存储对象。

对于安全来说, 这种区分非常重要。为了说明这一点, 不妨来看两种最主要的组织存储方式: 分段和分页。分段 (segmentation) 是将数据划分成逻辑单元, 每个段均有一个唯一的名字, 段内的数据项通过给出段名和适当的段内偏移地址来寻址。操作系统维护一个段表, 其中包括段名以及各个段在存储器的真实地址。Multics 操作系统使用分段来进行逻辑访问控制。

“+”分段是逻辑单元的一种划分, 它是实际安全策略的良好基础。

“-”段具有可变长度, 这使得存储管理较为困难。

分页 (paging) 将存储器划分为大小相同的页, 同样地址由两部分组成: 页号和页内偏移。

“+”分页很流行, 因为它允许高效的存储管理。Multics 的段实际上是分页的。

“-”分页对访问控制来说并不是一个很好的基础, 因为页不是逻辑单元。因此一个页可能包含要求不同保护的對象。

更糟的是, 分页可能会打开一个隐蔽信道。逻辑对象可以跨越边界存储, 当这样的对象被访问时, 操作系统在某个时候会要求一个新页, 这时便会产生缺页 (page fault)。如果缺页可以被查看到——就像大多数操作系统那样, 那么用户就获得了比正确访问请求结果更多的信息。

作为例子, 不妨来看这样一个口令方案。当用户输入一个口令后, 该口令被逐字符扫描, 并同保存在存储器的一个参考口令相比较, 当发现不匹配时拒绝访问。如果一个口令跨越页边界存放, 那么攻击者可从查看到缺页来推断出第一页上的口令片段已被正确猜测出来。如果攻击者可以控制将口令放在页的什么地方, 猜测口令就变得很容易了 (如图 5-9 所示)。第一步驟,

口令存放在内存中，它的第一个字符和其他字符存放在不同的页中。攻击者尝试第一个字符的所有值直到再次出现缺页，表明猜测是正确的。然后将口令在内存中重新排列，使得前两个字符和其他字符存放在不同的页中。攻击者已经知道了第一个字符，现在可以尝试第二个字符的所有值，直到再次出现缺页。通过不断的尝试，攻击者能够搜索到口令中的每一个字符。

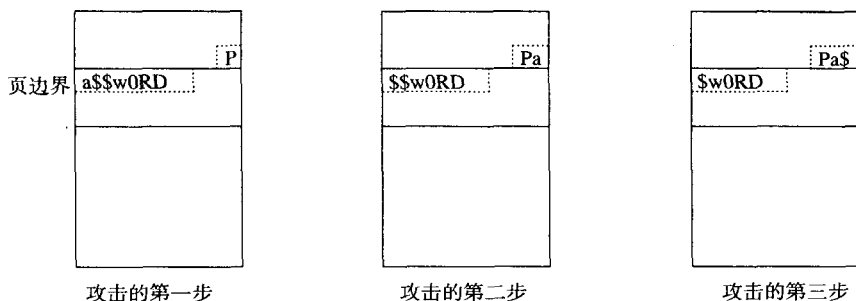


图 5-9 使用缺页作为隐蔽信道来猜测口令

安全编址

当你希望操作系统保护自身完整性，并将每个进程限制在各自的地址空间，那么任务之一就是控制存储器中数据对象的访问。这样，一个数据对象将在物理上被表示成存放在某个存储位置的比特集。访问一个逻辑对象最终被转换成机器语言级别上的访问操作。在这一层上，控制对存储位置的访问有三种办法。

- 操作系统修改从用户进程接收到的地址。
- 操作系统利用用户进程收到的相对地址构造有效地址。
- 操作系统检查从用户进程收到的地址是否在给定的界限内。

地址沙盒法是用第一种方法对应的实例。地址由一个段标识 (segment identifier) 和一个偏移量 (offset) 组成。当操作系统收到一个地址时，它会设置正确的段标识。图 5-10 显示了如何用两个寄存器操作来完成地址修改。首先，用地址与 `mask_1` 位“与” (AND) 操作清除段标识，然后用地址与 `mask_2` 位或 (OR) 操作将段标识设置成希望的值，`SEG_ID`。

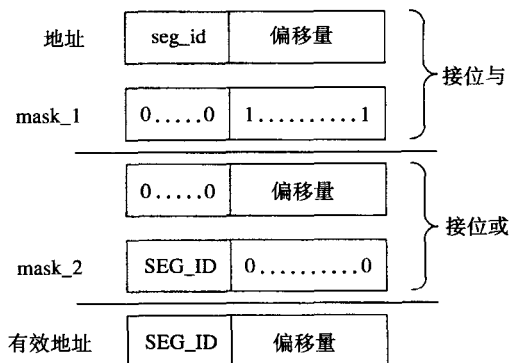


图 5-10 地址沙箱

第二种方法中，编址方式的巧妙使用避免了进程进入禁止的存储区域。如果需要更多关于编址方式的背景知识，可查阅 *Hennessy and Patterson (2002)* 或者其他有关操作系统或计算机体系

结构的书。在各种各样的编址方式中，相对编址应特别引起注意。

相对编址是用相当于某个给定的基地址的偏移量来指明的。

相对编址允许与位置无关的编码，这样一来，程序便可以存放在存储器的任何地方。存储器程序具有很大的灵活性，同时这也方便了栅栏寄存器(fence register)的使用。栅栏寄存器包含了分配给操作系统的存储区域的边界地址，用户程序中的地址被解析为相对地址(偏移量，位移量)，操作系统使用与栅栏寄存器相关的相对编址(基址寄存器编址)来计算有效地址(见图5-11)。使用这种方式时，只有操作系统空间以外的位置可被用户程序访问。操作系统采用类似的方法隔离分配给不同用户的存储区域。

这种方法可以通过用基址寄存器(base register)和边界寄存器(bounds register)来定义分配给进程的存储空间(见图5-11)，从而得到进一步的改进。我们甚至可以分别为用户的程序空间和数据空间进一步引入基址寄存器和边界寄存器。为了正确使用这种工具，处理器必须能够检测一个给定的存储空间存放的是数据还是程序代码。Motorola 68000 处理器就通过功能编码(function code)(图5-12)来支持这种隔离的。功能编码将处理器状态传给地址解码器，地址解码器能够使用这个信息在用户存储器和超级用户存储器之间进行选择，或者在数据和程序之间进行选择。

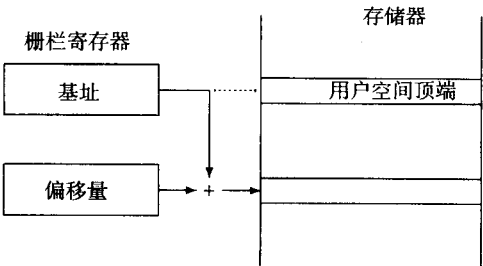


图 5-11 基址寄存器编址

FC2	FC1	FC0	
0	0	0	(未定义、保留)
0	0	1	用户数据
0	1	0	用户程序
0	1	1	(未定义、保留)
1	0	0	(未定义、保留)
1	0	1	监控数据
1	1	0	监控程序
1	1	1	中断接受

图 5-12 Motorola 68000 功能代码

经验教训

区分数据和程序的能力是非常有用的安全特性。它提供了保护程序不被修改的基础。

从一个更抽象的观点来看，内存已经被划分成了不同的区域。访问控制能够查看对象或程序的来源位置(location)。

经验教训

到此，微观世界中已经有了一个基于位置的访问控制的例子。在分布式系统或者计算机网络中，通常会需要这种访问控制。

很多指令集都没有检查它们的操作数类型的方法。在默认的情况下，类型信息可以在程序中由指定地址寄存器提供，这些地址寄存器可以在不同的内存访问操作中使用。这种解决办法需要合适的程序规则。

相反地，在一个有标记结构(tagged architecture)中，每个数据项都有一个用于表明其类型的标记。在执行前，CPU 会检查存储器中的值是否存在类型不符的情况。这些标记也会用于实施安全策略。在历史上，有标记结构曾经在理论上而不是在实际实现中盛行。有标记结构的少数例子就包括了 Burroughs B6500-7500 系统和 IBM 系统/38(Berstis et al , 1978)。(对于那些对计算机历史感兴趣的读者而言，Von Neumann(1993)在他的 1945 年的第一个有关电子数据计算机的报告中就描述了有标记结构)。图 5-13 显示了有标

标记	数据
整型
操作数型
位串型
.....
.....
.....

图 5-13 有标记的体系结构

记体系结构, 这个结构中显示了存储对象的类型。例如, 整型(INT)、位串型(STR)、操作数型(OP)。标记也可用于显示对存储位置可以采取的访问操作, 比如, 读、写或者执行。

5.5 深层阅读

基本的访问控制范例都归功于 *Lampson(1974)*。引用监控器最早起源于 *Anderson* 的报告(1972)。*Denning(1982)*全方位地描述了第一个多用户操作系统的计算机安全研究的结果。进一步的保护技术的调查可以在 *Landwehr(1983)*的著作中得到。在安全多用户操作系统设计中使用的的一个优秀的技术账户可参见 *Gasser(1988)*中得到, 该书已脱销, 不过可以在 <http://www.acsac.org/secshelf/book002.html>上查阅。这本书包括了这个领域很多有用的技术报告信息。

*Sibert, Porras and Lindell(1996)*讨论了用 Intel 80x86 处理器建立安全内核的事宜。*Wahbe et al.*, (1993)则在著作中描述了地址沙箱以及有关的技术。

*MacKenzie and Pottinger(1997)*比较了有关安全计算机系统的早期历史。可信计算基(安全内核)在构建安全系统时是否合适的不同观点分别可以参看 *Blakely(1996, 反对)*和 *Baker(1996, 支持)*的著作。*Schneider(2000)*充分探究了引用监控器设计的理论基础。

5.6 练习

练习 5.1 智能卡中的微处理器常将整个的操作系统放入 ROM 中。当前, 这个处理方式有向微处理器转变的趋势, 部分的操作系统可以下载到 EEPROM(电可擦写可编程只读存储器)中。将操作系统放在 ROM 中有什么优点和缺点? 将操作系统移动到 EEPROM 中会对安全有什么影响?

练习 5.2 没有安全内核是否安全? 讨论有一个可信计算基的安全内核(如 TCB)的优点和缺点。

练习 5.3 搜寻将以下三种原则应用在安全系统建立中的例子: 责权分离、抽象数据类型和原子操作。(原子操作必须全部执行以保证安全性。如果它被中断, 系统可能会终止在不安全状态。)

练习 5.4 所谓的寄生病毒会感染可执行程序。通过区分程序和数据来帮助构建抵御这种病毒的防护的能力将如何实现?

练习 5.5 一些缓冲区溢出攻击会把它们希望执行的代码放入调用栈中。通过区分程序和数据来帮助构建抵御这种特殊类型的缓冲溢出攻击的能力将如何实现?

练习 5.6 反病毒软件扫描文件来对抗攻击。一个病毒如何截取对内存的读请求以及隐藏它的存在?

练习 5.7 考虑这样的一个系统: 该系统会把事件号写入它的审计日志, 并且会用一张表将这些事件号转换为消息。在日志文件条目中使用这种间接级别有什么潜在的优点? 又有什么潜在的危险?

练习 5.8 作为一个学习例子, 试查看 SE Linux 中类型实施是如何实现的。

第6章 UNIX 安全

到目前为止，我们已经分别学习了各种安全机制。在真正的实现中，它们是相互依赖的。比如，访问控制和身份认证就必须协同工作，缺少了任何一个都无法有效工作。因此，我们现在要将注意力转移到由操作系统提供的安全机制。UNIX 是我们的第一个例子，因为它为我们提供了一个在相当细节的层面上研究安全机制的机会。Linux 下的安全机制与其同类操作系统 UNIX 极为相似，因此这一章可以作为 Linux 下安全的入门。

目标

- 了解典型操作系统提供的安全特色。
- 介绍 UNIX 安全基础。
- 明白实际系统中，通用的安全原则是如何实现的。
- 在不断变化的环境中，分析安全管理的任务。

6.1 引言

操作系统包括了一些基本构件，比如，身份识别和认证，访问控制和审计，以提供一致的安全控制集合。一旦我们决定支持灵活而且“富有特色”的安全策略，那么安全机制就会变得极为复杂。在那些环境中，TCB 将会过于庞大以至于无法放入到小型安全内核中。在我们的分层模型中，目前所研究的是由操作系统层所提供的安全控制。当评估操作系统安全性的时候，以下问题可以指导分析。

- 已经实现了哪些安全特色？
- 这些安全特色如何管理？
- 安全特色在什么条件下才起作用？

在最商业化的操作系统中，有一种通用的安全控制组织模式。有关用户(主体)的信息被存储在用户账户中。授予给用户的任何特权都被存放在这个账户中。身份识别和认证验证用户身份，使得系统能够将用户和由用户启动的进程联系起来。有关资源(对象)的许可能够由系统管理器或者资源所有者设置。在决定允许或拒绝访问请求时，操作系统可以参考用户身份、用户特权以及对象许可设置。

安全性不仅要解决对未授权行为的阻止，同时也要解决对未授权行为的检测。我们必须面对这样的事实，攻击者可能会找到绕过保护机制的方法。后备措施必须要跟踪用户已经执行的操作以便能够研究安全分支或者跟踪可能的攻击。因此，必须要求操作系统记录和保护与安全相关的审计日志(audit log)。

最后，如果不能正确使用操作系统的安全特色，那么操作系统最好的安全特色也将是毫无价值的。系统必须在一个安全状态下启动，因此系统的安装(installation)和配置(configuration)都是十分重要的问题。不完善的默认设置可能会导致重大安全缺陷。操作系统是高度复杂并且不断进化的软件系统。因此，随时都可能发现新的漏洞，或者排除已有的漏洞，或者发布新的版本。机警的系统管理者必须紧跟当前操作系统的开发。

我们将从以下几个方面来对操作系统的安全性框架进行描述：

- 主角、主体和对象。
- 访问控制。
- 审计、配置和管理。

这一章将验证 UNIX 操作系统的安全特性。由于其设计历史的原因，UNIX 的可靠性以及安全性并没有得到广泛的承认，参考 *B. F. Miller, Frederiksen et al. (1990)* 的著作。但是，UNIX 确实提供了安全特色，如果这些特色使用得当将会相当有效。而且人们对 UNIX 安全性的看法也发生了相当大的变化。不同的 UNIX 和 Linux 版本在技术实现以及安全控制实施方式上可能存在差异。本章中所用到的命令和文件名均采用了典型的使用方法表示，但这仍旧可能与特定系统有所不同。为了试图统一 UNIX 安全性，POSIX 1003 系列标准定义了 UNIX 系统通用接口。POSIX 1003.6 解决了安全机制问题。

这一章并没有打算完整的介绍 UNIX 安全性或者给出一个安全地建立 UNIX 系统的指导。我们只是局限于展示 UNIX 安全性的基础知识，并且重点突出大家关注的安全特性。

UNIX 安全体系结构

在大多数操作系统将安全体系结构 (security architecture) 解释为如何强化安全性以及将安全相关的数据存放于何处的同时，UNIX 却有着一段版本分化与融合的历史。这是现实的真实反映。每当有新的需求出现时，安全特色就被加入系统中，而不是沿袭最初的设计目标。

UNIX 最初是为网络环境下的小型多用户计算机所设计的，而后又扩展到了商用服务器，再而后又回缩到了 PC 机。如同因特网一样，UNIX 最初是为了友好环境而设计的，比如实验室或者大学，而且其安全机制较弱。随着 UNIX 的发展，新的安全控制被加入到系统中，已有的安全控制得到了加强。在实现新的安全特色时，设计者以尽可能少地与现有 UNIX 结构发生冲突为方针来指导设计。UNIX 的设计哲学是假定系统安全是由熟练的管理员来管理的，而不是有一般的用户管理的。因此，安全管理更多的是依靠脚本形式或者命令行形式的工具。

6.2 主角

主角就是所谓的用户标识符 (UID) 和组标识符 (GID)，UID 和 GID 是 16 位的数字。某些 UID 有特殊的意义，因操作系统的不同而不同，但是超级用户 (根) 的 UID 总是 0。UID 的例子见表 6-1。

表 6-1 UID 例子

-2	nobody
0	root
1	daemon
2	uucp
3	bin
4	games
9	audit
567	tuomaura

6.2.1 用户账户

有关主角的信息被存储在用户账户 (user account) 以及主目录中。用户账户被存放于 /etc/passwd 文件中。该文件中的表项格式如下：

用户名:用户口令:UDI:GID:ID 字符串:主目录:登录 shell

用户名 (user name) 是一个最大长度为 8 的字符串。在登录系统时，用户名用作身份识别而不用用于访问控制。UNIX 不会对具有同一个用户标识符的用户进行区分。口令是被加密存储的 (参见 6.3.1 节)。标识域包括用户的全名。最后两个域描述了用户的主目录 (home directory) 以及用户成功登录之后可用的 shell 类型。更为详细的用户设置是在用户主目录的 .profile 文件中定义的。当用户登录时，系统所执行的动作是 /etc/passwd 文件中定义的。用 cat /etc/passwd 或者 less /etc/passwd 可以显示口令文件，文件中的表项如下：

dieter: RT.QsZEEsxT92: 10026: 53: Dieter Gollmann: /home/staff/dieter: /usr/local/bin/bash

6.2.2 超级用户(根)

在每一种 UNIX 系统中,都有一个具有特殊权限的用户。这个用户拥有一个值为 0 的用户标识符,其用户名通常为根(root)。根账户是操作系统为了完成类似于登录这样的必要任务,记录审计日志或者访问 I/O 设备而使用的。

对于可以执行几乎所有操作的超级用户而言,所有安全检查均被关闭。比如,超级用户能够切换到其他任何用户。超级用户能够改变系统时钟。超级用户能够找到方法绕过施加于其上的少数限制。比如,超级用户不能写只读的文件系统,但是他可以卸载该文件系统,然后再以可写方式挂载该文件系统。超级用户不能解密口令,因为 crypt 是一个单向函数。

6.2.3 组

所有的用户都属于一个或多个组(group)。将用户安排在组中更便于执行访问控制决策。比如,可以将所有允许访问 E-mail 的用户安排到 mail 组中,将所有操作者都安排到 operator 组中。每个用户都属于一个基础组(primary group)。基础组的 GID 被存放在/etc/passwd 文件中。文件/etc/group 包括了所有组的清单,文件中表项的格式如下:

组名:组口令:GID:用户列表

比如,这个表项:

infosecwww: *: 209: chez, af

告诉我们组 infosecwww 的口令被禁用了,其 GID 为 209,而且有两个用户:chez 和 af。表 6-2 列出了具有特殊意义的组 ID。

在 System V UNIX 中,用户一次只能属于一个组。当前组可以通过 newgrp 命令来改变。用户可以自由地切换到他们所从属的其他组中。如果用户试图切换到他们所不从属的其他组中, newgrp 命令就会弹出来要求用户输入口令,如果组口令(group password)正确,用户就被给予暂时的成员资格。在 Berkeley UNIX 中,用户可以存在于一个以上的组中,因此不需要 newgrp 命令。

表 6-2 特殊组 ID

0	system/wheel
1	daemon
2	uucp
3	mem
4	bin
7	terminal

6.3 主体

主体是进程。每一个进程都有一个进程标识符(process ID)。新进程是通过 exec 或 fork 来创建的。每一个进程都同一个真实 UIDGID(read UID/GID)以及有效 UID/GID 相关联。真实 UID 是从父进程继承而来的。通常,它是登录用户的 UID。有效 UID 是从父进程或者从正在被执行的文件继承而来的(参见 6.5.1 节)。POSIX 兼容版本也预留了一个保留 UID/GID(saved UID/GID)。下面的例子说明了真实 UID/GID、有效 UID/GID 以及 UNIX 登录进程的作用。

进程	UID		GID	
	真实 UID	有效 UID	真实 GID	有效 GID
/bin/login	root	root	system	system
用户 dieter 登录, 登录进程验证用户名和口令, 更改 UID 和 GID:				
/bin/login	dieter	dieter	staff	staff
登录进程执行用户登录 shell:				
/bin/bash	dieter	dieter	staff	staff
用户通过 shell 执行 ls 命令:				

/bin/ls	dieter	dieter	staff	staff
---------	--------	--------	-------	-------

用户执行 su 命令, 以 root 身份启动一个新的 shell:

/bin/bash	dieter	root	staff	system
-----------	--------	------	-------	--------

6.3.1 登录和口令

在 UNIX 中, 通过用户名对用户进行身份识别, 并通过口令对用户进行认证。当系统启动之后, 登录进程被启动并作为 root 运行。当用户登录之后, 该进程验证用户名和口令。如果验证成功, UID/GID 被切换为用户的 UID/GID, 同时用户的登录 shell 被执行。根用户的登录可以被限制在/etc/ttys 中指定的终端上。用户上次登录时间被记录在/user/adm/lastlog 文件中, 可以通过 finger 等命令查看。

在许多 UNIX 系统中, 口令长度被限制在 8 个字符内。有一些工具通过阻止使用弱口令能够有效地支持口令选择。口令用 crypt(3) 算法进行加密(确切地说, 是进行 hash 运算)。该算法使用全 0 块作为输入并使用口令作为加密密钥, 连续 25 次使用略经修改的 DES 算法进行加密。加密后的口令被存储在/etc/passwd 文件中。

当用户的口令域为空时, 用户不必提供口令便可登录。当用户的口令域以星号打头时, 用户不能登录系统。因为对明文口令施以任何一种单向函数都不可能得到这样的值。这是一种禁止用户账号的通用方法。

通过 passwd(1) 命令可以改变口令。首先, 系统要求输入以前的口令, 以预防别人在用户没有锁住屏幕就离开电脑时更改用户口令。由于在输入口令的时候, 字符是不会显示在屏幕上的, 因此用户被要求两次输入新口令, 以确保想要输入的口令就是实际输入的。在更改口令之后, 可以重新登录或者使用 su(1)(set user) 命令来确认更改后的效果。

6.3.2 影子口令文件

安全敏感的 UNIX 版本提供了进一步的口令安全规定。/etc/passwd 是通用可读的, 因为它包含了为许多程序所需要的用户账户信息。因而, 攻击者可以拷贝口令文件, 然后用离线字典搜索口令攻击。为了去除这一漏洞, 口令被存储在影子口令文件(shadow password file)中, 比如 /.secure/etc/passwd, 该文件只能被根用户访问。该文件也可以用于实现口令老化和自动账户锁定。文件表项由 9 个域构成:

- 用户名。
- 口令。
- 上次更改日期。
- 口令更改之间的最小天数。
- 口令有效的最大天数。
- 提前多少天通告用户口令即将失效。
- 用户在线天数。
- 登录失效的日期。
- 保留域。

口令加盐是另一种减缓字典攻击的方法。salt 是一个 12 位的随机数, 被恰当地加入到口令中, 并且以明文方式保存。

6.4 对象

访问控制对象包括文件、目录、存储设备以及 I/O 设备。由于访问控制的原因, 所有的对象

都被统一视为资源(resource)。资源被组织成树型结构的文件系统。

6.4.1 i 节点

目录中的每一个文件表项都是一个指针，指向名为 i 节点数据结构。表 6-3 给出了 i 节点中与访问控制相关的域。每一个目录都包含了一个指向其自身的指针，即文件‘.’，以及指向其父目录的指针，即文件‘..’。每一个文件都有其属主。通常，属主就是创建文件的用户。每一个文件都属于一个组。

在不同版本的 UNIX 中，新建文件要么从属于其创建者组，要么从属于其目录组。

在讨论 i 节点中的域之前，我们先用 ls-l 来考察一个目录，结果如下：

```
-rw-r--r-- 1 diego staff 1617 Oct 28 11: 01 adcryp.tex
drwx----- 2 diego staff 512 Oct 25 17: 44 ads/
```

分析可得到的信息：

- 第一个字符指出了文件类型：“-”代表文件，“d”代表目录，“b”代表块设备文件，而“c”代表字符设备文件。
- 接下来的 9 个字符指出了文件的许可(file permission)，下面还要讨论。
- 后面的数字域是链接计数器(link counter)，用于计算该文件上的链接(link)数目。
- 接下来的两个域是属主名和文件组。
- 然后是文件字节大小。
- 时间和日期是 mtime，即上次修改时间。ls-lu 显示 atime，即上次访问时间。ls-lc 显示 itime，即上次修改 i 节点的时间。
- 最后一个域是文件名。ads 后面的“/”表示是一个目录。文件名被存储在目录中，而非 i 节点中。

文件许可(许可位)分为三组，分别定义了属主(owner)、属组(group)以及其他用户(other)(也叫做 world)的读、写和执行的权限。“-”表示权利未被授予。因此，rw-r--r--表示属主有读、写权限，属组以及其他用户只有读权限。rwx-----表示属主有读、写以及执行权限，属组和其他用户没有任何权限。

通过将这 9 个许可位分为 3 组，UNIX 系统中文件许可也可以通过八进制数表示。每一种访问控制权限均由一个比特位表示，如果这个访问位被设置的话，就被赋予了访问权限。表 6-4 中显示了这些数字。权限的联合就是响应数字的总和。比如，许可位 rw-r--r--等价于 644。许可位 777 就给定属主、属组，以及其他用户所有的访问权限。

表 6-3 i 节点中与访问控制相关的域

mode	文件类型以及访问权限
uid	拥有文件的用户
gid	拥有文件的组
atime	访问时间
mtime	修改时间
itime	i 节点更改时间
block count	文件大小
	物理位置

表 6-4 访问许可位八进制表示

400	由属主读
200	由属主写
100	由属主执行
040	由属组读
020	由属组写
010	由属组执行
004	由其他人读
002	由其他人写
001	由其他人执行

6.4.2 默认许可位

UNIX 工具，比如编辑器或者编译器，在新建文件的时候，通常使用 666 作为默认许可位，而在新建程序的时候，通常使用 777 作为默认许可位。这些许可位可以通过 umask 进一步调整。

umask 是一个包含三个数字的八进制数，它指定了应该被保留的权利。所以，umask 777 拒绝所有的访问，umask 000 就不做任何限制。敏感的默认设置有：

022 属主的所有许可，属组和其他人的读和执行许可

037 属主的所有许可，属组的读许可，其他人无许可

077 属主的所有许可，属组和其他人无许可

实际的默认许可由 umask 屏蔽(masking)UNIX 工具的默认许可而得到。umask 按位取反再和默认许可按位取与。比如，默认的许可位 666 和 umask 077，由 666 AND NOT(077)，我们经计算得到 600。这将给予文件所有者读、写权限，而其他权限都被拒绝。可以通过如下命令更改 umask：

```
umask [-S] [mask]
```

这里，-S 用于标注符号模式。如果没有给出掩码，该命令显示当前的 umask。

/etc/profile 里的 umask 定义了一个系统范围内的默认设置。这些默认设置可以被用户重新设置，具体方式是设置用户主目录里的 umask。根据 UNIX 的具体安装情况，具体位置可以在/etc/profile，.profile，.login 或者 .cshrc 里。和其他操作系统不一样，UNIX 不可能为目录定义单独的默认许可，也不可能让文件从目录继承许可。

用拷贝命令 cp 创建文件时，文件许可是从 umask 中获得的。用重命名命令 mv 创建新文件时，文件将会保持已有的许可许可位。

6.4.3 目录的许可

每一个用户都有一个主目录，比如/home/staff/dieter。可以使用 mkdir 命令建立子目录。用户必须拥有正确的文件许可才能在指定的目录中创建新的文件和目录(见表 6-5)。

- 读许可允许用户查找目录中有哪些文件，比如执行 ls 或者其他类似的命令。
- 写许可允许从目录中添加或者删除文件。
- 执行许可必须将目录设为当前目录才能打开其中的文件。如果用户知道目录中存在某个文件，才可以打开该文件，但用户不能用 ls 命令查看该目录中的内容。

因此，为了访问自己的文件，用户必须要在目录中有执行许可。为了阻止别的用户读你的文件，既可以设置对应的文件访问许可位，也可以阻止对目录的访问。你必须要有对目录的写和执行许可才能够删除目录。你并不需要有关文件的任何许可，它甚至可能属于别的用户。针对该特性，需要提醒系统管理员：

如果你试图将一个永久文件安装到一些用户的目录里，那么麻烦真的就降临了。

早期 UNIX 版本的一个遗留问题是粘滞位。它最初的作用是使某个程序在首次执行后继续留在虚拟存储器中。这样，系统就避免了把那些需要经常访问的程序代码传送到页面空间中。而今，粘滞位用于限制删除文件的权限。比如，作业队列通常是完全可写的，这样任何人都可以添加文件。然而，在这种情况下，每个人也都可以删除文件。设置了粘滞位的目录中的文件，只有在用户是文件的属主，目录的属主并且拥有写权限的情况下或者用户为超级用户的情况下，才能被删除或重命名。

用 ls-l 查看一个带有粘滞位的目录时，将用 t 而不是 x 表示执行许可。

6.5 访问控制

访问控制是基于主体(进程)和对象(资源)属性之上的。标准的 UNIX 按照属主、属组、其他人三种权限来管理资源。超级用户不受这种访问控制的限制。UNIX 按照统一的方式来对待所

有资源，它不区分文件和设备。UNIX 按照以下顺序来检查访问许可位：

- 如果用户的 uid 表明用户是文件的属主，属主访问许可位就决定了用户是否可以获得访问权限。
- 如果用户不是文件的属主，但是用户的 gid 表明用户在的组拥有该文件，组许可位决定了用户是否可以获取访问权限。
- 如果用户既不是文件的属主，又不属于文件的属组，那么其他人访问许可位决定用户是否可以获取访问权限。

因此，通过设置访问许可位可以使得文件属主的访问权限少于其他用户。这听起来让人有些吃惊，但却是很宝贵的经验教训。对于任何访问控制机制而言，都必须知道不同访问控制规则的检查顺序。

6.5.1 设置 UID 和 GID

一起回到我们最感兴趣的一个话题：受控调用。在执行某些系统调用时，UNIX 要求超级用户特权，比如只有根用户才能监听受信端口 0 ~ 123，但又不能给普通用户以超级用户的状态。所以必须找到能够满足这两个要求的方法。UNIX 采用的方法是提供 SUID(设置用户标识符)和 SGID(设置组标识符)程序。这样的程序与属主或属组的有效 UID 或有效 GID 一起运行，拥有暂时的或者受限制的访问权限。这些访问权限通常情况下是不赋予普通用户的。当 ls-l 命令显示一个 SUID 程序时，属主的执行许可位是 s 而不是 x：

```
-rws--x--x 3 root bin 16384 Nov 16 1996 passwd
```

当 ls-l 显示一个 SGID 程序时，属组执行许可是 s 而不是 x。在八进制表示的许可中，位于属主、属组及其他人前的 4 个八位字节以及粘滞位一起用于表示 SUID、SGID 程序和目录(见表 6-5)。

经常有这样的情况，如果根用户是 SUID 程序的属主，这时运行这个程序的用户就会在执行期间获得超级用户的状态。重要的 SUID 程序有：

/bin/passwd	更改口令
/bin/login	登录程序
/bin/at	提交批处理作业
/bin/su	更改 UID 程序

表 6-5 八进制表示的 SUID 及 SGID 程序

4000	在执行时设置用户标识符
2000	在执行时设置组标识符
1000	设置粘滞位

在这里我们照惯例要提出警告。既然用户拥有了运行时的 SUID 程序属主的特权，只应该让这个程序做属主程序准备做的事情。这一点对于属主是 root 的 SUID 程序尤为重要。一个可以改变 SUID 程序执行行为的攻击者，比如可以中断程序执行的攻击者，不仅可以在攻击时执行需要超级用户权限的行为，还可以改变系统使其在别的情况下同样具有这样的权限。这方面的危险发生在有用户交互的 SUID 程序中。一个特别的漏洞是 shell 逃逸(shell escape)。它使用户能以超级用户权限运行 shell 命令。除非实在必要，程序不应该被设置为 SUID。系统管理员应该十分注意监控 SUID 程序的完整性。Garfinkel, Spafford and Schwartz(2003)给出了两个使用 SUID 程序成功攻击的实例。

6.5.2 更改许可

文件的许可位可以用 chmod 命令修改，这个修改命令的执行者只能是文件的属主或者是超级用户。这个命令的格式如下：

```
chmod [-fR] absolute_mode file
```

指定所有访问许可位的值

<code>chmod [-fR] [who] +permission file</code>	添加许可
<code>chmod [-fR] [who] -permission file</code>	删除许可
<code>chmod [-fR] [who] =permission file</code>	重置指定许可位

在绝对模式(absolute mode)中,文件许可由一个八进制数直接指定。在符号模式(symbolic mode)下,当前文件许可将被修改。参数 who 可以使用如下值:

- u 改变属主许可。
- g 改变属组许可。
- o 改变其他人许可。
- a 改变所有人许可。

许可参数可以使用如下值:

- r 读许可。
- w 写许可。
- x 文件执行许可,目录的搜索许可。
- X 只有在文件是一个目录或者至少一个执行位被设置的情况下的执行许可。
- s SUID 或者 SGID 许可。
- t 保存文本许可(设置粘滞位)。

-f 选项用于抑制错误信息, -R 选项用于在当前目录的所有子目录中递归地执行该命令。

程序的 SUID 许可可以这样设置:

<code>chmod 4555 file</code>	设置 suid
<code>chmod u+s file</code>	设置 suid
<code>chmod 555 file</code>	清除 suid
<code>chmod u-s file</code>	清除 suid

GUID 许可不使用 u, 而使用 g 选项。

chown 命令可改变文件的属主, chgrp 改变文件的属组。chown 是不受欢迎的 SUID 程序的潜在来源。用户可以建立一个 SUID 程序, 然后把它的属主改为 root。为了避免这样的攻击, 有些版本的 UNIX 只允许超级用户运行 chown。别的 UNIX 版本允许用户对他们自己的文件使用 chown, 不过这时就关闭了 SUID 和 SGID 位。类似的考虑适用于 chgrp。

6.5.3 UNIX 访问控制的不足

文件只能有一个属主和属组。许可只能控制读、写以及执行访问权限。因而, 其他所有的权限, 比如关机权限或者创建新用户的权限, 都被映射到基本的文件访问许可。除了读、写和执行之外的其他许可都必须由应用实现。通常, 很难利用 UNIX 的访问控制机制实现更为复杂的安全机制。在这个方面, UNIX 安全性更多的是依赖于人一机标尺的机器端(图 6-1)。

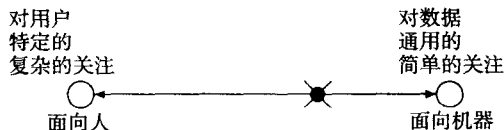


图 6-1 人一机标尺上的 UNIX 安全

6.6 一般安全原则的实例

这部分将以 UNIX 为例, 来看看一般的安全原则是怎样实现的。

6.6.1 受控调用的应用

被多个用户访问的敏感资源，比如网络服务，可以使用结合了属主、许可位和 SUID 程序概念的受控调用来施加保护。

- 创建一个名为“Web server”的新的 UID，它拥有资源以及需要访问该资源的所有程序。
- 只给这个资源的属主以访问许可。
- 把所有访问资源的程序设置为 SUID 程序。

经验教训

当心保护过度。如果用户需要某个文件来完成他们的工作，而你却拒绝用户对该文件的直接访问，那么你必须通过 SUID 程序为用户提供对该文件的间接访问。有缺陷的 SUID 程序可能比精心设计的访问许可位给予用户更多的权限。当资源和 SUID 的属主是类似 root 一样的特权用户时，更是如此。

在这个案例中，我们看到了一个在安全系统设计里经常使用的技术实例。抽象属性在系统中是由数据结构表示的。该数据结构又被其他安全机制以不同的目的重用。UID 被用来代表系统中的真实用户。现在，UID 被用来进行一种新的访问控制，在此，它并不代表任何真实用户。

6.6.2 删除文件

第二个最感兴趣的话题：逻辑存储结构与物理存储结构的比较。特别是，如果我们从文件系统中删除了一个文件会怎样呢？这个文件还会以某种形式继续存在吗？

UNIX 有两种拷贝文件的方法。cp 命令用于创建一个一模一样但是独立的副本。除了 cp 命令，还有一些命令 link 和 ln，它们只是创建了文件名和指向原来文件的指针，同时增加了原来文件的链接计数器(link counter)。新、旧文件共享文件内容。如果原来的文件用 rm 或者 rmdir 删除了，那么它就从其父目录中消失了，但是它的内容和拷贝依然存在。这样，用户可能认为他们已经删除了文件，但是实际上它仍然存在于其他目录中，用户仍然拥有它。如果你想要彻底删除一个文件，超级用户必须运行 ncheck 列出文件的所有链接并且删除这些链接。如果一个文件在被别的进程打开后由属主删除，这个文件在此进程关闭它之前会一直存在。

一旦文件被删除，分配给该文件的存储空间就可以被重用了。然而，除非这些存储区域被再次使用，否则它们仍旧保留着文件的内容。为了避免类似的存储残留(memory residue)，在删除文件之前，应该使用全零或者适合于存储媒介的模式抹去这些内容。尽管文件还没有被完全删除，一些高级文件系统(如，defragmenter)可能会移动文件从而在磁盘上留下更多的文件拷贝。

6.6.3 设备保护

下一个问题仍然与逻辑存储结构和物理存储结构的区别有关。UNIX 像对待文件一样地对待设备。这样，对存储器或者对打印机的访问都可以像访问文件一样设置许可位。设备文件用 mknod 命令建立。该命令只能由 root 执行。在/dev 目录下，通常可以发现如下设备：

/dev/console	控制台终端
/dev/mem	主存储器的映像设备(物理存储器的映像)
/dev/kmem	内核存储器映像设备(虚拟存储器的映像)
/dev/tty	终端

如果攻击者能够访问存放文件和目录的存储设备，那么他们就能绕过设置于文件和目录之上的安全控制。如果存储器设备对其他人的读写许可位被设置，攻击者就能浏览或者修改存储器上的数据，而不会受存储器中文件的访问许可的约束。因此，几乎所有的设备都应该禁止其他

人的读写。

命令如进程状态命令 `ps` 显示内存的使用情况，因此需要对内存设备的访问许可。把 `ps` 定义为 `root` 的 `SUID` 程序可使其获得必需的权限。不过，`ps` 的威胁在于可能会让攻击者得到 `root` 权限。更好的解决办法是让 `ps` 成为 `SGID` 程序，让 `mem` 组拥有内存设备。

`tty` 终端设备是另一个有趣的例子。用户登录的时候，一个终端文件就被分配给这个用户，用户就成了本次会话中该文件的属主（终端文件空闲的时候，它的属主是 `root`）。让这个文件对其他人可读写会带来诸多方便，因为这样一来，用户就可以接收来自其他用户的消息。不过，这也带来了安全隐患。别人现在就可以监视进出该终端的流量。这里面可能包含用户的口令。他们可以向用户终端传送命令，比如，重新设定一个功能键，再让不知情的用户执行这些命令。在一些系统中，智能终端会自动执行这些命令。这就给攻击者用其他用户的权限提交命令提供了机会。

6.6.4 改变文件系统的根

在沙盒（`sandbox`）中，通过限制可疑进程可以实现访问控制。对沙盒外部对象的访问将被阻止。在 `UNIX` 中，改变根（`change root`）的命令 `chroot` 限制了未授权用户可以访问的文件系统。该命令只能由 `root` 执行。

```
chroot <目录> <命令>
```

上述命令执行时，根目录从 `/` 变成了“目录”。只有新根目录下的文件是可访问的。如果采用这种方案，就必须确定用户程序可以找到所有它们需要的系统文件。这些文件的预期在为 `/bin`，`/dev`，`/etc`，`/tmp` 或者 `/usr` 等类似目录。必须新的根目录下建立有着相同名字的新目录，然后拷贝或者链接到原来目录里的文件。

6.6.5 挂接文件系统

当用户拥有不同的安全区域并且要从别的区域向用户的系统里引入对象的时候，可能需要重新定义这些对象的访问控制属性。

`UNIX` 文件系统把存在于不同物理设备上的文件系统链接在同一个根（用“`/`”表示）下面。这通过 `mount` 命令实现。在网络环境中，远程文件系统可以从别的网络节点中挂接。同样，允许用户通过他们自己的软盘挂接文件系统（`automount`）。

如果你是安全专家的话，警钟就要开始长鸣了。挂接上的文件系统可能包括各种不受欢迎的文件，比如，隐藏在攻击者目录中指向 `root` 的 `SUID` 程序。一旦文件系统被挂接，攻击者就可以通过运行此类程序获得超级用户的状态。允许直接访问内存的设备文件也很危险，在这里许可已经被设置，从而攻击者可以访问这些文件。所以，命令中常带有 `-r` 标志和一些选项。

```
mount [-r] [-o 选项] 设备 目录
```

`-r` 标志表示只读挂接，而选项里包括：

<code>nosuid</code>	在挂接的文件系统中关闭 <code>SUID</code> 和 <code>SGID</code> 位
<code>noexec</code>	挂接文件系统上的二进制文件不能被执行
<code>nodev</code>	在挂接的文件系统中不能访问块或者特殊字符设备

当然，不同版本的 `UNIX` 实现 `mount` 命令的不同选项。

经验教训

`UID` 和 `GID` 都是局部描述符。在不同的 `UNIX` 系统上，它们不必按照同样的方法进行解释。在挂接远程文件系统的时候，用户可能错误地解释这些描述符。因此，你应该在网络上使用全局

唯一的描述符。

6.6.6 环境变量

环境变量是由 shell 所保存，通常被用来配置工具程序的行为。表 6-6 列出了 bash shell 的部分环境变量。进程默认从其父进程继承环境变量，而且执行其他程序的程序可以将被调用程序的环境变量设置为任意值。

由于 SUID/SGID 程序的调用者控制了这些程序被赋予的环境变量值，所以就会出问题。攻击者可能会将环境变量值更改为危险值尝试并以此控制程序的执行流程。此外，许多库和程序都是以一种含糊的、未公开的方式被环境变量所控制的。比如，攻击者可以将 IFS

设置为异常值并以此来绕过保护机制。该保护机制可以过滤对 SUID/SGID 程序的危险输入(更多内容参见第 14 章)。作为一种对策，SUID/SGID 程序可以清除整个环境并将必需的环境变量的子集重新设置为安全值。

经验教训

继承你不想要的或者不了解的东西可能会造成安全问题。

6.6.7 搜索路径

我们最后要讨论的话题是从“错误”位置启动的程序。UNIX 用户通过 shell(命令行解释器)和操作系统交互。为了方便，用户在执行一个程序时，可以只键入它的名字而不用说明它在文件系统里的完整路径名(full pathname)。shell 会依照用户主目录的 .profile 文件中 PATH 环境变量所指定的搜索路径(searchpath)查找程序(用 ls -a 来查看主目录里所有文件，用 more .profile 来查看这个文件)。当找到一个包含所给名字的程序的目录时，搜索结束，并且执行那个程序。典型的搜索路径如下所示：

```
PATH=.: $HOME/bin: /usr/ucb: /bin: /usr/bin: /usr/local: /usr/new: /usr/hosts
```

在这个例子里，搜索路径里的目录用“:”分隔。第一个条目“.”是当前目录。于是就可能以这样的方式插入特洛伊木马：给木马取一个和现有程序一样的名字，把它放在一个比存放原始程序的目录更早搜索到的目录中。

为了防止这样的攻击，就要完整路径名来启动程序，比如，用/bin/su 而不是 su。当然，需要确定当前目录不在根用户执行的程序的搜索路径上。

6.6.8 包裹层

目前，我们所提到的访问控制和审计机制并不复杂。它们和操作系统安全的传统一致，集中在对资源的访问控制上。还有可能通过巧妙地使用基本的访问控制机制实现对中间层的控制。此外，我们还可以修改 UNIX 本身来实现这个目标。这里的难点是找到这样一个 UNIX 组件，我们可以通过修改它来增加有用的安全机制，同时不会影响到操作系统的其他部分。考虑到 UNIX 的复杂性，这个任务并不轻松。

TCP 包裹层非常精彩地说明了这样的设计方法。UNIX 的网络服务，比如 telnet 和 ftp，是按

表 6-6 Bash shell 的环境变量

PATH	shell 命令的搜索路径
TERM	终端类型
DISPLAY	显示名称
LD_LIBRARY_PATH	搜索对象和共享库的路径
HOSTNAME	UNIX 主机名
PRINTER	默认打印机
HOME	主目录路径
PS1	默认提示
IFS	用于分隔命令行参数的字符

照如下的原则建立的。inetd 守护进程监听外来的网络连接请求。当连接建立后, inetd 启动相应的服务程序, 然后再转去监听新的连接请求。这个守护程序被叫做超级服务器 (super-server), 因为它为许多服务程序工作。inetd 守护进程有一个配置文件将服务(端口号)映射到程序上。该配置文件格式如下:

service	type	protocol	waitflag	userid	executable	command-line
(服务	类型	协议	等待标识	用户 ID	可执行程序	命令行)

比如, telnet 的条目是:

```
telnet stream tcp nowait root /usr/bin/in.telnetd in.telnet
```

当 inetd 接收到一个服务请求, 它会先查看配置文件, 然后创建一个新进程以运行指定的可执行程序。这个新进程的名字将被更改为在命令行字段中给出的名字。

一般情况下, 可执行文件的名字和命令行字段中给出的名字相同。这个冗余开启小伎俩的方便之门。把 inetd 守护进程指向一个包裹层程序(wrapper program), 而不是原来的可执行程序。使用进程的名字来记住原来的可执行程序, 在包裹层里执行完安全控制之后, 原来的可执行程序就被执行了。在我们的例子里, 配置文件中 telnet 的条目可被替换为:

```
telnet stream tcp nowait root /usr/bin/tcpd in.telnet
```

被执行的程序现在成了 /usr/bin/tcpd, 这就是包裹层可执行程序。运行包裹层程序的进程仍然叫做 in.telnet。在包裹层程序里, 用户可以进行所有想要做的访问控制或者审计控制。在原来的应用中, 包裹层被用来进行 IP 地址过滤(参见第 13 章)。因为包裹层知道它所在的目录, 如 /usr/bin, in.telnet, 知道它的名字, 如 in.telnet, 因此它可以调用原来的服务程序, 如 /usr/bin/in.telnet。在用户看来没有任何差别, 他们接受的服务和以前一样。

经验教训

在计算机科学里, 再增加的间接层异常强大。在安全上, 它可以用来攻击或者保护系统。通过在 inetd 守护进程和服务程序之间插入 TCP 包裹层, 可以在不改变守护进程和服务程序源代码的情况下增加安全控制。

这个例子的优点在于它的通用性。同样, 同样的机制可以用来保护一整套 UNIX 网络服务。

经验教训

TCP 包裹层结合了一个基本的设计原则(受控调用)和一个完美的技巧, 使得在不改变这些服务程序的情况下增加新的安全机制成为可能。在更新已有的系统以加入新的安全特性的时候, 这便是非常理想的情况。

6.7 管理问题

我们将快速地讨论一些有关管理 UNIX 系统操作安全的问题。

6.7.1 管理超级用户

操作系统使用 root 账户来执行自己基本的任务以及其他系统管理任务。因为超级用户的权限极大, 所以它们是 UNIX 主要的弱点。获得超级用户身份的攻击者能够有效地接管整个系统。因此, 必须十分谨慎地控制对超级用户身份的访问。

攻击者可以通过将其自身的 UID 修改为 0 从而成为超级用户。因此, 必须对 /etc/passwd 和 /etc/group 文件进行写保护。为了减小这种威胁的影响, 可以划分管理者的职责, 比如创建 uucp 或者 daemon 用户来处理网络。这样的话, 即使某个用户受到威胁, 也不会影响到全部用户。系统管理者不应该将 root 用作个人账户。在必要的时候, 可以通过键入 /bin/su(不需要指定用户

名)切换到 root。操作系统不会引用存放于另一个目录中的 su 程序。在审计日志中记录 su 命令的使用以及使用 su 命令的用户账户。

6.7.2 可信主机

在友好的环境下,尽管有大量不同的主机被访问,但对用户进行一次身份认证已经足够了。在 UNIX 中,可信主机(trusted host)支持这种操作模式。来自可信主机的用户不需要口令认证就可以直接登录。只要他们在双方的主机上都有相同的用户名就可以了。/etc/hosts.equiv 文件中描述了机器的可信主机。用户主目录中的 .rhosts 文件描述了用户的可信主机。

随着主机数量的增加,任务变得单调乏味,因此用户名必须在主机之间同步。(有一些特定厂商的配置工具可供使用。)一旦主机被列入/etc/hosts.equiv 文件,该主机上的所有用户都拥有了访问权限。很难对例外用户进行设置。

6.7.3 审计日志与入侵检测

一旦系统安装好并投入运行之后,其安全机制就应该阻止非法用户的操作。然而,保护机制并不够或者存在缺陷。不恰当的安全设置可能无法有效地保持系统持续运行。因此,在安全漏洞发生时或者发生后,采取进一步的安全机制检测安全违背或者其他相关的安全事件很有必要。一些安全相关的事件被自动记录在 UNIX 的日志文件中。

- /usr/adm/lastlog—记录用户上次登录时间;该信息可用 finger 命令查看。
- /var/adm/utmp—记录账户信息,可用 who 命令查看。
- /var/adm/wtmp—记录用户每次登录或登出时间;该信息可用 last 命令查看。为了防止该文件占据所有的存储空间,每隔一段时间该文件会被自动删减。
- /var/adm/acct—记录所有执行过的文件;该信息可用 lastcomm 命令查看。

这些文件确切的名字以及位置,在用户的 UNIX 系统上可能有所不同。记账功能,由 accton 命令打开,也能用于审计目的。find, grep, ps, users 命令可用于进一步查看 UNIX 系统。

检查上述日志文件中与安全相关的事件清单。这些事件绝大多数与用户相关,因此日志条目应该包括引发事件的进程的 UID。那么审计又是怎样被 SUID 程序所影响的呢?这样的程序是属主的 UID 运行的,而不是以用户的 UID 来运行程序的。因此,日志表项也应该包括进程的真实 UID。

经验教训

用户标识是一个安全属性,使用它有两个目的,访问控制和问责性。并不总是有可能同时为了两个目的使用这个属性。只要 UID 对应于真实的用户,基于许可的访问控制和审计就相互依存。一旦创建了一个利用 SUID 或者 SGID 程序保护对资源的访问的特殊的用户标识,就同时拥有了一个在审计方面有特定应用的属性。

6.7.4 安装与配置

操作系统生命周期中的一个关键点就是安装。操作系统有许多安全特性和影响安全的特性,其中一些没有被公开。从历史的观点上来说,默认设置有利于顺利地安装和操作,但同时它给予了维护工程师和系统管理员太多的特权。正确的做法是:像限制每一个别的用户一样限制系统管理员,并分清系统管理员和安全管理员的角色。复杂并且未完全公开的安全特性可能会让系

统的安装变得非常困难，所以制定安全策略很急迫。UNIX 本身并未给系统管理员的工作带来便利。

- 系统管理员必须熟知所有与安全相关的文件以及安装以后如何对危险的默认设置做修改。
- 系统安装后，安全相关的参数用标准的 UNIX 编辑命令定义。对资源访问的许可设置在更接近操作系统的层次而非应用层。比如，通过编辑类似/etc/passwd 的文件创建用户。用以下的命令保护 passwd 程序：

```
chmod 4750 /bin/passwd chgrp staff /bin/passwd
```

- 在审计系统时，要使用 UNIX 搜索命令。比如用下面的指令扫描没有口令的账户：

```
awk -F: 'length($2) < 1 print $1' < /etc/passwd
```

搜索 SUID 和 SGID 的命令：

```
find-type f(-perm 2000 -o -perm 4000) -exec ls -ld {};
```

(有经验的 UNIX 用户以能够编写这类命令而引以为荣，但一般用户认为很难以这种方式管理系统。)

- 可以通过简单的自主访问控制支持访问控制策略。结构化保护可以基于组成员关系 (group membership) 和使用被禁止的账户来实现。

有许多附加的 UNIX 安全产品可以用来管理安全特性以及检查当前安全状态。两个流行的检测工具是 COPS (*Framer and Spafford, 1990*) 和 SATAN。它们可搜索已知的安全漏洞，比如弱口令、文件和目录的不安全许可设置，或者畸形的配置文件。系统管理员可以使用这些工具来检测他们所管理的系统中的漏洞。不过，这种做法不太普遍，因为它们可以被攻击者用来实现几乎相同的目的。

6.8 深层阅读

这一章对 UNIX 安全进行了概览。如果你想要更全面地了解这些内容，有关这方面的书籍很多，比如 *Curry (1992)*, *Ferbrache and Shearer (1992)*, *Garfinkel, Spafford and Schwartz (2003)* 的著作。如果你需要有关特定 UNIX 版本的信息，请查询制造商提供的文档和系统提供的在线文档 (使用手册)。在 *Samalin (1997)* 的著作中讨论了多级安全 UNIX 系统。在网上可以找到有关增强安全性 (Security Enhanced, SE) Linux 的信息。来自计算机紧急响应组 (CERT) 的安全建议可以帮助你关注最新的安全漏洞信息。

6.9 练习

练习 6.1 查看安全相关命令的在线文档。找到在/etc/passwd 中你自己的条目，并查看你的文件和目录的许可权设置。

练习 6.2 在你的目录下创建一个子目录，放入一个带有短消息的 welcome.txt 文件。将这个子目录的许可属性设置为属主可执行，然后：

- 用 cd 命令使该子目录成为当前目录。
- 列出该子目录的文件清单。
- 显示 welcome.txt 文件的内容。
- 在该子目录中创建 welcome.txt 文件的一个拷贝。

再分别对该子目录设置读许可和写许可，重复上述练习。

练习 6.3 哪条 UNIX 命令可以列出你的目录下其他人可写的文件？

练习 6.4 你怎样保护一个终端设备(tty)不被其他用户访问?

练习 6.5 你能够在 VSTa 能力框架(4.7.4 节)下,通过 UID、GID 和许可权,来捕获 UNIX 的访问控制吗?

练习 6.6 应用 UNIX 安全机制实现 Chinese Wall 模型和 Clark-Wilson 模型?

练习 6.7 你应该怎样建立备份过程来减少安全漏洞?

练习 6.8 就你看来,UNIX 系统安全的长处和弱点是什么?就此写一篇 1000 字左右的报告。

第7章 Windows 2000 安全

UNIX 的访问控制将所有的对象统一地视为资源。相反, Windows 2000 的访问控制可以被裁剪为单个的对象类型。因此, 我们将以 Windows 2000 为例来展示怎样才可能构件一种更加细密的访问控制方法。需要强调的是我们并没有以某个特定的版本作为参考, 而是基于 *Swift et. al. (2002)* 对 Windows 访问控制可能的扩展的描述。同其他操作系统一样, Windows 的安全性是一个动态目标, 而 Windows 的设计原理更加稳定。

目标

- 介绍 Windows 2000 安全的基本概念。
- 说明如何通过重定向实现更易于管理的访问控制。
- 说明如何管理目录中的访问权限的继承。
- 从基于身份的访问控制到基于代码的访问控制。

7.1 引言

我们不会完整地介绍 Windows 2000 的安全性或者给出如何充分利用其安全特性的指导。我们的主要目标是对比 UNIX 与 Windows 的安全机制从而突出为大家所关注的安全特性, 这也强调了计算机安全的根本性问题。我们将关注于原理并粗略地讲解其他方面, 比如用于管理安全特性的图形界面。Windows 2000 获得了通用标准 EAL4 证书(参见 10.6 节), 该事实也说明了 Windows 在安全方面付出的努力。

7.1.1 体系结构

图 7-1 给出了 Windows 系统的体系结构。正如在 UNIX 中一样, 用户模式(ring 3)与内核模式(ring 0)之间被隔离开了。硬件抽象层(Hardware Abstraction Layer, HAL)提供了对计算机硬件的接口。核心操作系统服务, 即 Windows 执行部件, 运行于内核模式。执行部件包括了负责访问控制的安全引用监控器(Security Reference Monitor)。

用户程序通过应用程序接口(application program interface, API)调用来操作系统的服务。上下文切换以及从 ring 3 到 ring 0 的转换都是由本地过程调用(Local Procedure Call)来处理的。设备驱动程序(通常是第三方产品)运行于内核模式。由于它们自身代码中的漏洞会被攻击者利用进而控制 Windows 系统, 比如缓冲超限, 因此设备驱动程序也是与安全紧密相关的。运行于用户模式的安全子系统(security subsystem)部件如下:

- 登录进程(Log-on Process, winlogon): 在用户登录时, 对用户进行认证的进程。
- 本地安全权威(Local Security Authority, LSA): 在用户登录时, 检查有用户账户并创建访问令牌(access token), LSA 还负责审计功能。
- 安全账户管理员(Security Account Manager, SAM): 维护用户账户数据库。在本地用户认证期间, LSA 将使用该数据库。

口令以 hash 形式存储于 SAM 中。hash 函数的特性表明无法从存储于 SAM 中的值得到口令。此外, 口令 hash 值数据库已被加密。

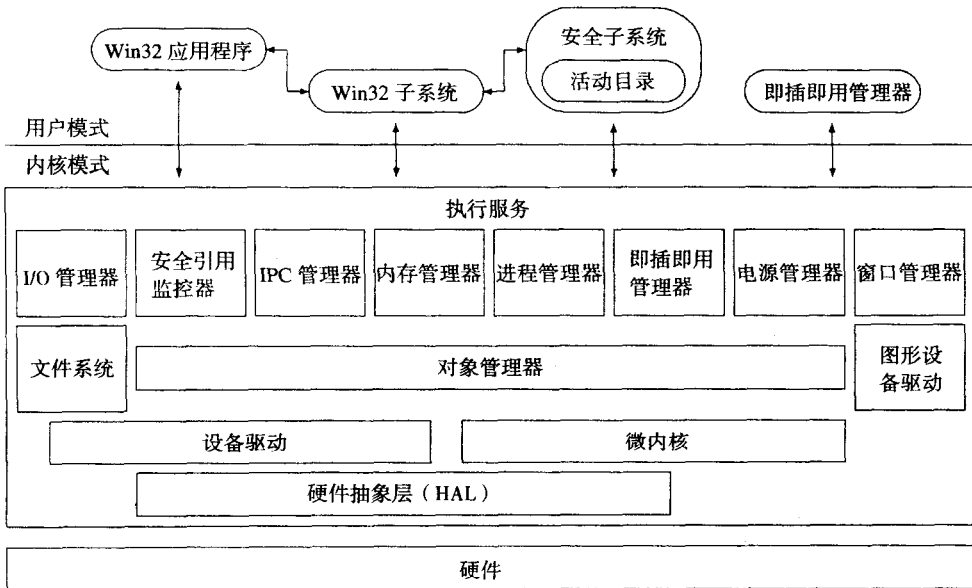


图 7-1 Windows 系统体系结构(微软公司产品屏幕截图, 已获得微软公司许可)

7.1.2 注册表

注册表(registry)是 Windows 配置数据的中央数据库。注册表中的表项称为键(key[⊙])。用于修改注册表的工具称为注册表编辑器(Registry Editor, 即 Regedit.exe 或 Regedt32.exe)。也可以用注册表编辑器来查看注册表。在注册表的顶层, 有 5 个预定义的重要键。

- HKEY_CLASSES_ROOT: 包括了文件扩展名关联, 比如可以让 .doc 文件由 Word 软件来处理。
- HKEY_CURRENT_USER: 当前登录用户的配置信息。
- HKEY_LOCAL_MACHINE: 与本地计算机相关的配置信息。
- HKEY_USERS: 包括了系统中已经加载的所有活动用户的描述文件。
- HKEY_CURRENT_CONFIG: 在系统启动时, 本地计算机所用到的有关硬件描述文件的信息。

注册表蜂房(hive)是一组键、子键以及键值。与安全相关的注册表蜂房有:

- HKEY_LOCAL_MACHINE\SAM
- HKEY_LOCAL_MACHINE\Security
- HKEY_LOCAL_MACHINE\Software
- HKEY_CURRENT_CONFIG
- HKEY_USERS\DEFAULT

在注册表中, 可以根据用户的需求对系统进行裁减, 并设置默认的保护策略。当然, 攻击者也可以通过修改注册表项来改变操作系统的行为。比如注册表的键指向操作系统搜索特定可执行文件的位置。在 Windows 中, 这些位置被称为路径(path)。如果这类键的许可设置较弱, 比如任何用户都拥有写权限, 那么攻击者就可以通过修改路径来插入恶意软件。因此必须保护注册表数据的完整性。从那些不被用来管理系统的所有计算机中删除注册表编辑器是第一道安全防

⊙ 注意不要与加密“密钥”相混淆。——译者注

线。一些安全相关键不应该由注册表编辑器修改，而应该通过特定的工具来修改。

在对注册表键设置访问控制策略的时候，你将不得不面对一个进退两难的问题：过强的保护和使用的方便是不能够同时满足的。在设置策略时，一个潜在的危险在于未定义的键。比如，考虑下面这个键表明哪些用户和组可以远程访问注册表。

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurePipeServers\Winreg

如果这个键存在，当用户请求远程访问注册表时，将访问该键。如果该键不存在，就不需要会对远程访问执行任何检查。这会使得对注册表的远程访问就和本地访问一样。

7.1.3 域

单独的 Windows 计算机通常由用户在本地进行管理。然而，在组织机构中，必须采用一种更加结构化的方法来进行系统管理以及安全管理。作为计算机网络中的用户，你不愿意在访问另一台计算机上的服务和资源时重复地输入口令。作为计算机网络的管理员，你可能不想对每一台计算机进行单独配置。Windows 2000 使用域 (domain) 来实现一次签到以及集中式的安全管理。

域是共享公用用户账户数据库和安全策略的计算机集合。域可以构成层次结构。用户不在每台计算机上创建账户，但需要域账户。

在域中，要有一台服务器充当域控制器 (DC, domain controller)。接着，其他计算机将加入到这个域中。域管理员 (domain admin) 创建并管理这个域中的用户和组。域控制器权威拥有与用户口令相关的信息，并且能在用户向其他实体认证其自身的时候充当第三方可信当事人。在这里，就已经做出了集中式认证 (口令管理) 服务的设计决策。

域可以拥有多个域控制器，可以在任何一个域控制器中进行更新操作，并且通过多主复制模型传播更新。当今，分布式服务被作为用于获取更好的性能的设计决策。

7.1.4 活动目录

对象被组织在活动目录 (Windows 2000 中的目录服务) 中。活动目录可以被看作一棵由特定类型对象所构成的树 (图 7-2)。容器是可以包含其他对象的对象。活动目录可以通过添加新的对象类型以及为已存在的对象类型添加新属性动态扩展。因此，用户可以根据自己的需要来裁剪对象类型。每一种对象类型 (object type) 都有一个特定的属性以及一个唯一的 GUID (globally unique identifier)。此外，每一种属性都有自己的 GUID。在图 7-2 中，描述了一个具有姓名 (name)、E-mail、地址 (address) 以及房间 (room) 的“员工”对象和一个具有模式 (mode) 和房间 (room) 的打印机对象。

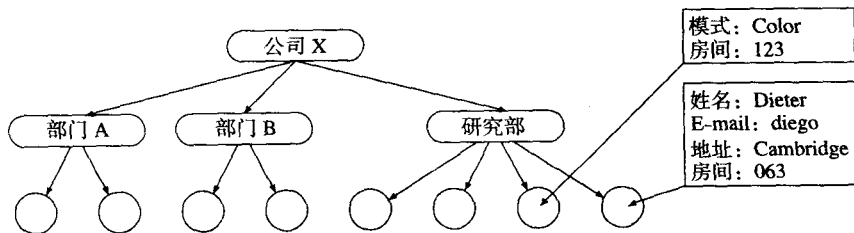


图 7-2 目录树

从逻辑上来说，不同类型的相关对象可以放在相同的容器中。这对结构中的资源管理是有用的，因为不需要为不同的对象类型建立不同的结构，还因为可以使用目录结构来为容器定义通用的访问策略并让容器内的对象继承策略 (参见 7.3.3 节)。

7.2 访问控制——组件

Windows 的访问控制比典型文件系统的访问控制更加复杂。访问控制可以应用于文件、注册表键以及活动目录对象等。Windows 2000 的结构化策略是组、角色以及继承。下一节将讲解 Windows 2000 的主角、主体以及对象，以及在哪里可以找到访问控制规，如何评估这些规则。

7.2.1 主角

主角是安全策略中的活动实体。它们是可以被允许或拒绝访问的实体。在 Windows 2000 中，主角可以是本地用户、别名、域用户、组或者是机器。主角有一个人们可读的名字(用户名)以及机器可读的描述符号，安全标识符(security identifier, SID)。

本地用户是由本地安全权威所创建的用户。本地主角(Local principal)是在本地管理并且只对本地计算机可见的。例如：本地系统，即操作系统以及本地用户。本地用户或别名的可被人们读懂的名字具有如下形式：

principal = MACHINE\principal,

比如，“TUHH-688432\Administrators”。可以通过以下命令在命令行^①下显示用户和别名：

- net user
- net localgroup

域主角(domain principal)由域控制器上的管理员管理，比如域用户和域管理员别名，它们对域中的所有计算机都是可见的。域用户、组、别名或者机器的可被人们读懂的名字具有如下形式：

principal@ domain = DOMAIN\principal,

比如，“diego@ europe. microsoft. com = EUROPE\diego”。可以通过向命令行中输入如下命令显示域用户、组以及别名：

- net user /domain
- net group /domain
- net localgroup /domain

存在通用的主角(universal principal)，比如 Everyone 别名。有关主角的信息存储在账户和用户描述文件中。本地账户存储于注册表中(位于 HKEY_USERS 键下)。域账户存储于 DC 中，但是被本地缓存了。账户描述文件存储于文件系统中，位于“\ Documents and Settings \”目录下。一些预定义的主角不需要存放在任何地方。

1. 组和别名

SID 是一个单独的主角。(全局)组是一个由 DC 管理的 SID 构成的集合。组有自己的组 SID，因此组可以被识别。组的用户可以使用分配给该组的特权和许可。组构成了控制中间层。对象的许可分配给了组。通过将用户变成组的成员，用户就被给予了对对象的访问权限。

别名(本地组)是由 DC 或本地管理的用户和组 SID 构成的集合。别名不可识别。别名被用于实现逻辑用户。应用程序开发者可以参考别名 Student。在部署时，一个合适的 SID 被分配给该别名。别名并非 4.6.3 节中定义的“角色”。

2. 安全标识符

SID 的格式为 S-R-I-SA-SA-SA-N，其中

S：字符 S。

① 单击“开始”，然后单击“运行”，输入 cmd，再键入命令。

- R: 更正号(当前为 1)。
 - I: 身份权威(48 比特)。
 - SA: 子权威(32 比特)。
 - N: 相对描述符, 在权威名空间中是唯一的。
- 下面的清单给出了典型的主角以及它们的 SID。

- Everyone(World): S-1-1-0。
- SYSTEM: S-1-5-18; 操作系统以 S-1-5-18 运行于本地机器上; 对于域中的其他机器而言, 该机器有一个单独的特定域 SID。
- 管理员: S-1-5-21- <本地权威>-500; 用户账户是在操作系统安装期间创建的。
- 管理员组: S-1-5-32-544; 具有管理员特权的内置组; 最初它只包括管理员账户。
- 本地管理员: S-1-5-21- <域权威>-512; 全局组是域中所有计算机管理员别名的一个成员。
- 来宾: S-1-5-21- <权威>-501; <权威>字段是一个 96 比特的唯一的机器或域描述符, 该描述符是 Windows 或 DC 安装时创建的。

SID 是在创建用户账户或者修改已经到期的账户时创建的。由于在创建过程中使用了伪随机输入(时钟值), 如果删除了账户然后用与以前相同的参数创建账户, 将不可能得到相同的 SID。因此, 新的账户不会保留给予原有账户的访问许可。

在创建域的同时, 会为该域创建一个唯一的 SID。在一台工作站或服务器加入一个域的时候, 它会收到一个包括了这个域的 SID 的 SID(计算机使用它们的 SID 来检查它们是否位于同一个域中)。由于 SID 不能被修改, 要想在不同域之间移动 DC 并不是一件简单的管理工作。必须完全重装机器并使其逻辑上成为一台“新的”机器从而接收新的 SID, 这样才能成为新的域中的控制器。

7.2.2 主体

主体就是操作系统中的活动实体。在 Windows 2000 中, 主体指进程和线程。进程或线程的安全凭证存放在访问令牌(access token)中。

SID 用作身份识别和认证属性。令牌也包括了分配给这些 SID 的所有特权的联合。新对象的默认值包括了诸如属主 SID、组 SID 和 DACL(在 7.2.3 节中解释)等。各种实体都包括了登录会话 ID 以及令牌 ID。令牌中的某些字段是只读的, 其他字段可以修改。一个新的进程在一定的限制条件

用户安全标识符
组和别名安全标识符
特权
新对象的默认值
杂项

可获得其父进程的访问令牌。即便成员资格或者特权被撤销, 令牌也不会改变。因为进程可以提前决定它是否有权访问一个特定的任务, 所以这就提高了系统的性能和稳定性。

1. 特权

特权控制着对系统资源的访问。特权由语法名字而唯一标识, 比如 SeTcbPrivilege, 特权还具有一个用于显示的名字, 比如“作为操作系统的一部分”。特权被分配给每一台计算机上的用户、组以及别名。它们作为本地唯一的标识符(LUID)被缓存于令牌中。特权不同于访问权限, 后者控制对安全对象(securable object)的访问。典型的特权有:

- 备份文件和目录。
- 生成安全审计。
- 管理和审计安全日志。
- 接管文件和目录的所有关系 R。
- 绕过遍历检查。
- 允许计算机和用户账户作为可信代表。

- 关闭系统。

2. 用户认证 - 交互登录

Windows 用户可以通过用户名和口令进行认证, Windows 也支持其他认证方式, 比如使用智能卡进行认证。在登录计算机的时候, 用户通过按下安全警示序列 (Ctrl + Alt + Del) 来启动认证过程。该序列调用 Windows 操作系统的登录屏幕并在键盘和登录进程 (winlogon.exe) 之间提供一条可信路径。登录对话框是由 GINA DLL (Graphical Identification and Authentication dynamic-link library, 图形识别动态链接库) 创建的。登录进程将 (以 SYSTEM 主角) 一直运行。

为了避免欺骗攻击, 在开始会话的时候一定要按 Ctrl + Alt + Del, 即使登录屏幕已经显示了也应该如此。安全警示键可产生对低级 Windows 函数的调用, 而这些函数不可能通过应用程序来复制。只有在计算机真正运行起来之后, 这一可信路径才是可见的。运行其他操作系统的计算机可以模拟 Windows 的登录屏幕来发动欺骗攻击 (Hadfield, Hatter and Bixler, 1997)。

Windows 提供了显示法律公告对话框的选项作为警告消息。在登录继续进行之前, 用户必须确认这一警告消息。接下来要求用户输入用户名和口令。登录进程获取到用户名和口令并传递给 LSA (lsass.exe)。对于本地登录而言, 本地 LSA 调用认证包 (authentication package) 来将用户名和口令与存储在账户数据库中的值进行比较。当找到一个匹配的时候, SAM 就将用户的 SID 和用户所属的组的 SID 返回给 LSA。域登录使用 Kerberos, 并且用户是由 DC 上的 LSA 认证的。然后, LSA 创建包括了用户 SID 和特权的访问令牌并将该令牌传递给登录进程。

3. 创建主体和网络登录

下一步, 登录进程在新的登录会话中以被认证用户 (主角) 的身份启动一个 shell (explorer.exe), 并且将访问令牌附着到该进程上, 见图 7-3。shell 创建的进程都位于同一个会话中。这些进程就是访问控制目的的主体。退出会终止会话以及会话中的所有进程。

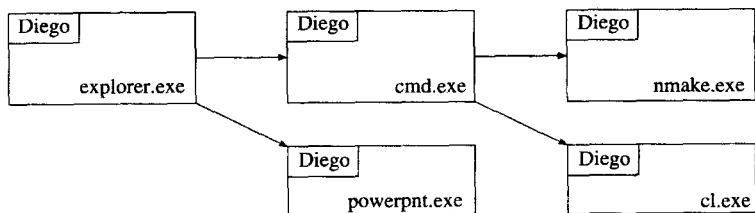


图 7-3 实例: 用户会话中的进程

进程可以通过调用 CreateProcess 创建新的本地主体 (进程)。新进程得到其父进程令牌的一个拷贝。每一个进程都有自己的令牌。在登录会话中的不同进程可以有不同的信任凭证。线程可以被分配不同的令牌。

在交互登录会话 (interactive logon session) 中, 用户的网络凭证, 比如口令, 都被缓存了起来。那么, 进程可以为其他机器上的用户创建网络登录会话 (network logon session)。网络登录会话通常不会缓存凭证。在 Windows 2000 中, 机器是主角并且在域中拥有一个带有口令的机器账户。因此, DC 也能够认证机器。

7.2.3 对象

对象是访问操作中的被动实体。在 Windows 2000 中, 执行对象 (executive object) 包括进程或线程, 而文件系统对象 (filesystem object) 包括文件或目录。其他对象包括注册表键和设备, 比如打印机。安全对象具有安全描述符 (security descriptor)。内置对象的安全描述符是由操作系统管理。私有对象的安全描述符必须由应用程序自己管理。创建安全对象可能是一件很枯燥的事情,

但是这样可以实现粒度更小的访问控制。

1. 安全描述符

安全描述符具有如下结构：

属主安全标识符
主组
自主访问控制列表
系统访问控制列表

属主安全标识符域表明了对象的属主。在创建对象的时候，它们就有了属主。属主通常是具有 READ _ CONTROL 和 WRITE _ DAC 许可的主角。属主关系也可以通过“接管对文件和其他对象的所有关系”(SeTake OwnershipPrivilege)。主组(Primary Group)是为了与 POSIX 兼容而引入的。自主访问控制列表(Discretionary Access Control List, DACL)定义了谁将被给予或拒绝对对象的访问。系统访问控制列表(System Access Control List, SACL)定义了该对象的审计策略。

2. 许可

许可是对某一对象进行特定操作的授权。单击文件，然后单击“属性”，最后单击“安全”，这样就可以查看文件许可。访问权限(access right)对应于可以施加于对象的操作。应用于大多数对象类型的标准访问权限(standard access right)有：

- DELETE：删除对象。
- READ _ CONTROL：对所有主、组、DACL 的读权限。
- WRITE _ DAC：对 DACL 的写权限。
- WRITE _ OWNER：对属主的写权限。
- SYNCHRONIZE：使进程等待一个对象进入置位状态。

在 Windows 2000 中，可以为每一类对象制定特定的访问权限。这样，访问权限就可以适应于应用程序的需求，但这样一来开发者必须潜在地记忆大量的特殊权限。为了解决这个问题，可以使用通用访问权限(generic access right)作为中间描述层。每一类对象都具有从通用访问权限到真实访问权限的映射，因此就没有必要记忆文件和目录的特定类型的访问许可。表 7-1 列出了文件的通用访问权限。通用访问权限有：

- GENERIC _ READ
- GENERIC _ WRITE
- GENERIC _ EXECUTE
- GENERIC _ ALL

表 7-1 文件和目录的通用访问权限映射

GENERIC _ EXECUTE	FILE _ READ _ ATTRIBUTES STANDARD _ RIGHTS _ EXECUTE SYNCHRONIZE
GENERIC _ READ	FILE _ READ _ ATTRIBUTES FILE _ READ _ DATA FILE _ READ _ EA STANDARD _ RIGHTS _ READ SYNCHRONIZE
GENERIC _ WRITE	FILE _ APPEND _ DATA FILE _ WRITE _ ATTRIBUTES FILE _ WRITE _ DATA FILE _ WRITE _ EA STANDARD _ RIGHTS _ WRITE SYNCHRONIZE

7.2.4 访问掩码

就其内部而言，访问请求中所要求的操作（请求的权限）以及所准许的访问权限都以访问掩码的形式给出。访问掩码是一个 32 位的值，其各个位的定义如下：

0 ~ 15	为请求所引用的对象定义的特殊权限
16 ~ 23	标准权限
24	访问系统安全，访问 SACL 必需条件
25	允许给予最大权限
26 ~ 27	保留
28	通用读写、执行
29	通用执行
30	通用写
31	通用读

设置 MAXIMUM_ALLOWED 位时，将给主体该安全描述符所允许的最大访问权限。所给予的最大访问权限按照 GrantedAccess 进行编码。标准权限的映射如右表所示。

16	删除
17	读控制
18	写_自主访问控制列表
19	写_属主
20	同步

7.2.5 扩展权限

在活动目录中，也可以为操作定义访问控制。这样的访问控制被叫做扩展权限（extended rights）。典型的例子是 Send-As 和 Receive-As，允许以指定的邮箱发送或接收邮件。扩展权限不是通过访问掩码来定义的，而是通过访问控制权限（controlAccessRight）对象对应的 GUID 来定义的。扩展权限列表不是固定的。研发者可以为惯用操作创建新的扩展权限。

7.3 访问决策

每一种对象类型都有一个对象管理器，它处理对象的创建并验证进程是否有权使用某个对象。比如，活动目录就是目录对象的对象管理器。对于访问控制而言，对象管理器调用 SRM 实现的访问决策函数。SRM（策略决策点），为对象管理器（策略执行点）返回一个是否的答复。

一般而言，访问控制决策要考虑请求访问权限的主体、要访问的对象以及所请求的访问权限。这三个参数并不是在所有的情况下都需要。主体的凭证，包括它的主角，都存储于其令牌中。对象的安全属性存储于其安全描述符中。主体所要求的访问操作以访问掩码的形式给出。

在创建对象句柄的时候，所请求的访问权限将与主体的令牌和安全描述符进行比较，而不是在访问的时候。因而，改变文件的 DACL 并不会影响已经打开的文件句柄。这一设计决策导致了更好的性能和更高的可靠性，因为所有的访问控制检查都已经在进程开始任务之前完成了。

Windows 访问控制可以以多种方式应用于不同的粒度和复杂度层面上。基于请求访问主角的访问控制被称为扮演，因为进程扮演了它的令牌中的用户 SID。这个方法粗糙但易于实现。扮演是操作系统中一个典型的概念，但是在应用层无效。以角色为中心的访问控制使用组和别名为进程分配适当访问权限。在以对象为中心的访问控制中，应用层对象会获得一个安全描述符。这种方法支持更严密的访问控制，但与此同时问题也复杂化了。

7.3.1 DACL

接下来的几节并不局限于特定的 Windows 版本，而是基于 Swift et. al. (2002) 提出的建议。安

全描述符中的 DACL 是一个访问控制表项(access control entry)清单。ACE 的格式如下：

- ACE 的类型决定了 ACE 将如何被访问控制机制使用。
- ACE 类型可以取如下值：
- ACCESS_ALLOWED_ACE;
 - ACCESS_DENIED_ACE;
 - ACCESS_ALLOWED_OBJECT_ACE(允许对对象、属性、属性设置以及扩展权限的访问);
 - ACCESS_DENIED_OBJECT_ACE(拒绝对对象、属性、属性设置以及扩展权限的访问)。

类型：肯定(允许)或否定(拒绝)
标志
ObjectType
InheritedObjectType
访问权限
主角 SID：ACE 应用的主角

ObjectType 是定义对象类型的 GUID。现在，应用程序可以在它们的访问请求中包括 ObjectType。对于一个给定的请求，只有具有一个匹配的 ObjectType 或不具有 ObjectType 的 ACE 才会被评估。比如，为了控制对对象属性的读/写访问，可以将属性的 GUID 放入到 ObjectType 中。为了控制对对象的创建/删除操作，可以将对象的 GUID 放入到 ObjectType 中。

下一个例子是网页目录的 ACE，它允许用户设置他们自己主页。这个策略应用于所有的用户，因此我们使用 PRINCIPAL_SELF_SID 作为占位符。在进行访问请求的时候，创建主页的应用程序会提供当前用户的 SID。

ACE1	
类型：	ACCESS_ALLOWED_OBJECT_ACE
对象类型：	Web 主页的 GUID
继承的对象类型：	用户账号对象的 GUID
访问权限：	写
主角 SID：	PRINCIPAL_SELF

以下 ACE 允许服务器应用程序在任何类型 RPC 服务容器中创建 RPC 终端。ACE 将被每一个类型 RPC 服务容器所继承。

ACE2	
类型：	ACCESS_ALLOWED_OBJECT_ACE
对象类型：	RPC 终端的 GUID
继承的对象类型：	RPC 服务的 GUID
访问权限：	创建子服务
主角 SID：	服务器应用程序

属性集

为了减轻管理工作，可以将对象类型的属性组织到属性集合中。我们只需要一个引用了属性集合的 ACE，而不是对象类型的所有属性 ACE。属性集合通过 GUID 来识别。在访问请求中，一系列的属性被传入引用监控器，对属性集合的一次检查将为每一个属性返回检查结果。其进一步的优点在于，对对象类型的修改并不会迫使我们更改 ACL。

7.3.2 决策算法

当主体请求访问对象时，SRM 取得主体的令牌、对象的 ACL 以及所请求的访问掩码以决定是否允许所请求的访问。它首先检查是否存在 DACL。如果没有 DACL，即所谓的 NULL DACL，那么不必进一步检查并允许该访问请求。

否则，决策算法接下来检查主题是否为对象的属主。如果所请求的访问掩码包括了 Read _

Control 或者 Write_DAC 请求, 则允许访问。如果不是这样, 就通过建立授权的访问掩码累积许可。对于每一个 ACE, 主体的 SID 将同 ACE 中的所有 SID 进行比较。以下三种情况都有可能:

(1) ACE 不包括匹配的 SID; ACE 将被跳过。

(2) ACE 包括请求访问权限的 AccessDenied(拒绝访问)SID; 访问将被拒绝并且不再进一步检查。

(3) ACE 包括了一个 AccessAllowed(允许访问)的 SID; 如果 ACE 中的访问掩码以及先前检查过的匹配的 ACE 中的访问掩码, 均包括了所请求的访问掩码中的所有许可, 访问将被允许并且不再进一步检查; 否则, 继续搜索。

如果 DACL 搜索完毕并且所允许的掩码不等于所请求的掩码, 访问将被拒绝。因此, 如果 DACL 为空, 那么访问将一直被拒绝; 如果不存在 DACL, 那么访问将一直被允许。

为了使否定的 ACE 优先于肯定的 ACE, 它们必须被放在 DACL 的顶端。正如同将在下一节中看到的那样, 为了实现更严密的访问控制也可以将否定的 ACE 放到肯定的 ACE 之后。

经验教训

通常, 操作系统将访问控制信息存储于不同的地方。了解检查执行的顺序非常重要。有时, 正如在 UNIX 中一样, 只有第一个匹配的访问控制表项会被考虑。但有时候也可能是这样的情况, 即相对于先前的表项而言, 后面的特定表项反而居于支配地位。最后, 必须知道如果没有找到匹配的表项操作系统将如何反映。

7.3.3 ACE 继承

在创建新的对象时, ACE 通常是从对象所在的容器(目录)继承而来。在活动目录中, 一个容器可能包括了不同类型的对象, 因此需要一个可选的继承策略。

Swift et al. (2002) 提出的机制是通过继承标志以及所继承的对象类型来控制继承的。继承标志表明了 ACE 是否已被继承。在创建新的对象时, ACE 是从容器继承而来的。当新对象创建的时候, 只有那些具有匹配的继承对象类型或者不具有继承对象类型的 ACE 被复制到它的 ACL。图 7-4 说明了这个问题。

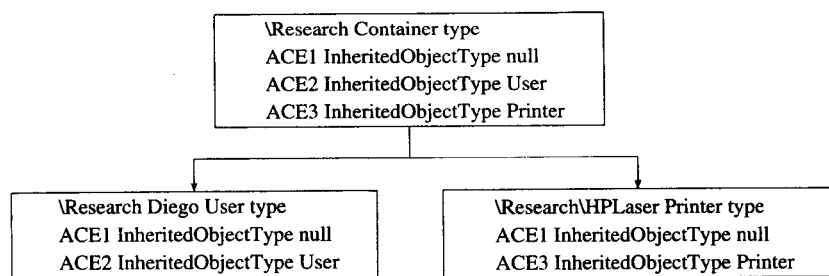


图 7-4 在容器内的特定类型的继承

对容器的修改不会立即对容器中已有的对象造成影响。这种策略称为静态继承(static inheritance)。如果你修改了容器的访问许可或者想修改、过滤其内容, 你必须运行传播算法(propagation algorithm)。该算法应该是指数级的。再次使用传播算法可能不会改变访问控制权限。而在动态继承中, 任何对容器的修改都将自动应用于容器内所有对象的地方, 因而与动态继承相比, 静态继承具有性能优势以及更易于预见的行为。

在设置容器的许可时, 通常不知道哪些对象最终将被放到容器中, 甚至可能不知道这些对象的属主, 但许多访问策略都给予对象的属主以特定的权限。因此, 在没有对所有者命名的情况下, 我们必须能够指出属主能够继承的权限。为了达到这个目的, Windows 2000 用了一个占

位符 SID。在创建新的对象时，在继承来的 ACE 中该 SID 被替换为属主的 SID。

1. 继承标志

继承标志进一步说明了如何继承 ACE。已经定义了标志如下：

- **INHERITED_ONLY_ACE**：ACE 只用作继承，而且访问检查机制将忽略该标志。
- **NO_PROPAGATE_INHERIT**：该标志只被下一代继承，而不会被进一步传播。
- **OBJECT_INHERIT_ACE**：该标志由所有不属于容器的子对象继承。
- **CONTAINER_INHERIT_ACE**：该标志被所有属于容器的子对象所继承。

2. 规则例外

在实际情况中，即便对于考虑得最周全的通用规则也会有例外。因此，我们需要一个既容易定义又便于应用规则，同时还要能够为这些规则定义例外的方案。由于 ACE 是按照它们在 DACL 中的顺序来进行评估的，将本地添加的表项放到继承而来的 ACE 之前就可以使得特定的表项优先于更为通用的表项。结果，来自近亲容器的 ACE 被放到来自远亲容器的 ACE 之前。这样，肯定性的 ACE 就可能出现在一个匹配的否定性 ACE 之前，如图 7-5。图中，ACE2 是直接为 Documents 容器中的 Letter_A 定义的，因而出现在了继承来的 ACE 之前。因为 ACE1 只继承了 Letter 类型的对象，所以它没有被 Invoice_A 所继承。

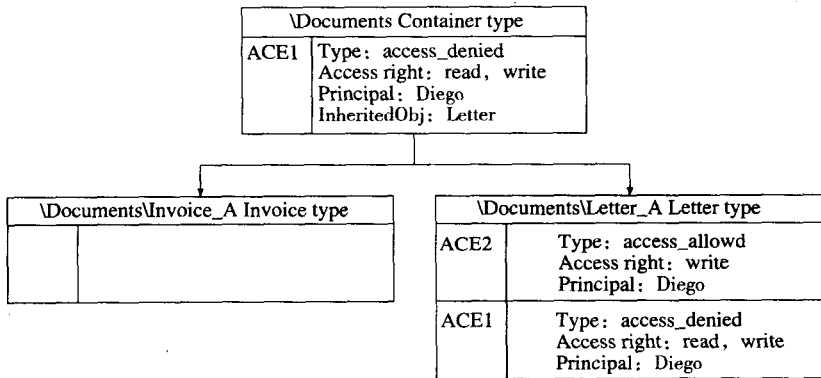


图 7-5 本地添加的 ACE 被放到继承来的 ACE 之前

将更加详细的表项放到更加通用的表项之前是一种创建例外的方法。Windows 2000 进一步实现了创建例外的机制。为对象的安全描述符设置 **SE_DACL_PROTECTED** 标志会阻止 ACE 继承，因而在设置对象许可时从一个干净的对象开始。在图 7-6 中所示，Letter_A 的安全描述符中设置了该标志，并且它的 DACL 只包含了一个本地定义的 ACE2。

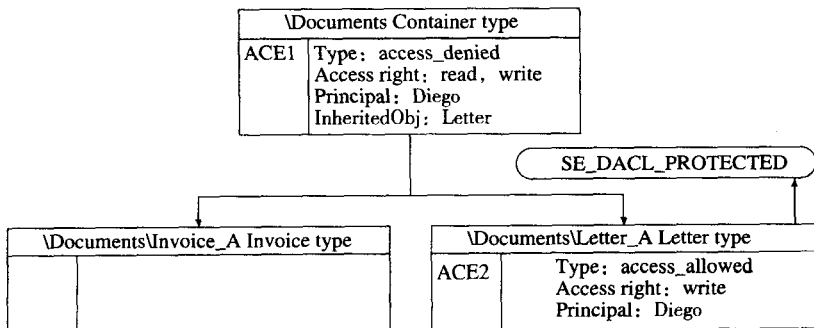


图 7-6 SE_DACL_PROTECTED 阻止了 ACE 继承

7.4 受限上下文

到目前为止，访问控制已经(显示地)涉及到了用户。用户登录时，进程的安全令牌中的 SID 被初始化，然后传入会话中的其他进程。我们可能需要采取一种补充方法并且也要控制某些程序能够做什么。目前，将这类程序称为“不可信代码”是很平常但不是特别的有帮助。这种称呼没有突出关键点。我们一直坚持好的安全设计实践并且遵循最小特权的原则。代码运行的时候，它只被给予了完成其任务所需要的许可。

在 Windows 中，可以使用受限令牌(restricted token)来实现基于代码的访问控制(code-based access control)。以受限令牌运行的进程就是受限上下文(restricted context)。受限令牌去掉了给定令牌中的特权。通过将组 SID 设置为 USE_FOR_DENY_ONLY，就可以禁止组 SID。在服务器线程模拟客户端角色的时候，比如在客户端访问令牌的上下文中运行时，这一特色很有用。为了方便起见，客户端令牌可能被给予了太多的权限[⊖]，而禁用 SID 提供了一种禁止给予客户端访问权限的方法。

向令牌添加受限 SID 后，只有在 SID 和受限 SID 都被允许访问时，具有受限令牌的进程才能获取访问权限。在图 7-7 中，一个具有受限令牌的进程请求访问具有不同 DACL 的对象。在图 7-7a 中，主角 SID Diego 和受限 SID MyApp 都具有读许可，因此允许访问。在图 7-7b 中，Admin 组 SID 被设置为只拒绝，所以 ACE3 将被跳过而且访问将被拒绝，因为主角并不具有其所请求的权限。在图 7-7c 中，尽管主角已经拥有了其所请求的权限，但受限 SID MyApp 没有读许可，因此访问仍将被拒绝。

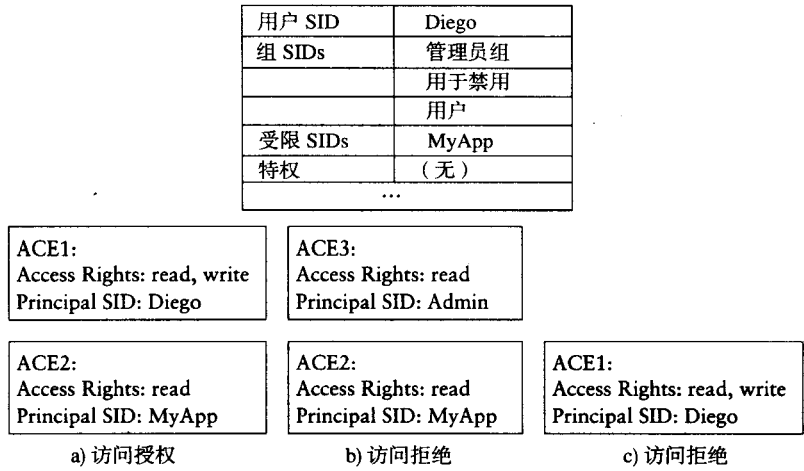


图 7-7 带有受限令牌的访问

为了限制程序的访问权限，我们创建一个代表该程序的受限 SID。在我们的例子中，我们有一个为 MyApp 创建的 SID。该 SID 被添加到程序所要访问的所有对象的 DACL 中。可以为每一个进程创建受限 SID，并将受限 SID 添加到程序所请求的资源(对象类型)和每一类对象类型的 ACL 中，或者添加到所要访问的主体的受限令牌中。

⊖ 面对客户端定义的输入和访问权限，服务器可能会执行危险的操作(困惑的代表问题)。

7.5 管理

我们将通过进一步评论 Windows 2000 的安全管理来总结这一章。

7.5.1 用户账户

有关用户的安全相关的信息都被 SAM 存储于用户账户数据库中。用域用户管理器 (User Manager for Domains) 工具可以修改用户账户, 并且可以在命令行中用网络用户的 `username` 命令来显示用户账户。在所有域中, 可以定义以下域。

- 用户名 (Username): 用户登录的唯一名字。
- 全名 (Full name): 拥有账户的用户名字。
- 过期时间 (Expiration date): 默认情况下, 账户没有过期日期。
- 口令时间 (Password dates): 上次修改口令的日期, 口令过期日期 (将当前口令设置为过期, 就可以强制用户在下次登录时修改口令), 口令可以被修改的时间; 也可以指明用户是否可以修改他们的口令。
- 登录时间和地点 (Logon hours and workstation): 可以指定用户可以登录的时间以及从哪个工作站登录。在登录时间到期后断开远程用户与服务器的连接, 这一设置决定了是否将用户从已有会话中剔除出去还是继续连接。在后一种情况中, 只阻止新的登录。
- 用户描述文件和登录脚本名 (User profile path and logon script name): 描述文件定义了用户的桌面环境, 比如程序组、网络连接、屏幕颜色等。登录脚本是一个批处理文件或可执行文件, 在用户登录的时候, 它会自动运行。
- 主目录 (Home directory): 你也可以指定主目录是否位于本机上或者在网络服务器上。
- 本地组和全局组 (Local and global groups): 用户所在的组。

7.5.2 默认用户账户

Windows 通过默认账户支持安全管理。你已经看到了 7.2.1 节的第一个例子。有三种类型的默认用户账户和组账户。

- 预定义账户是同操作系统一起安装的。
- 内置账户是同操作系统、应用程序以及服务一起安装的。
- 显式账户是在访问网络资源时显式创建的。

由操作系统创建的默认用户和组可以被修改, 但不能被删除。LocalSystem 账户是一个内置账户, 它用于运行系统进程以及处理系统级任务。用户不能登录到该账户, 但某些进程可以。

管理员 (Administrator) 账户和来宾 (Guest) 账户是在本地安装的预定义账户。管理员 (Administrator) 账户不能被删除或禁用。它拥有对文件、目录、服务以及设备的完全访问权限。尽管可以通过修改文件和目录的访问许可使管理员 (Administrator) 无法访问, 但是管理员 (Administrator) 仍旧能够修改访问许可并最终获得访问权限。默认情况下, 域管理员账户是管理员 (Administrator)、域管理员 (Domain Admins)、域用户 (Domain Users)、企业管理员 (Enterprise Admins)、计划管理员 (Schema Admins) 以及组策略创建者属主组 (Group Policy Creator Owners) 的成员。

在一个域中, 当系统第一次安全安装时, 主要使用本地管理员账户。一旦安装完成, 实际管理员会被添加到管理员组 (Administrator group)。因而, 单独的管理员权限更容易被撤销。

来宾 (Guest) 账户是为只是偶尔需要访问系统的用户而设置的。可以像给其他用户账户一样

给定这类用户一定的许可权限。当 Windows 2000 安装完成之后, 来宾 (Guest) 账户就被禁用了。

内置组 (built-in group) 已经预定义了用户权限和许可, 并且在授予用户权限的时候提供了一个间接层。用户可以成为这样一个组的成员来获得标准的访问权限。典型的内置组例子是管理员组、备份操作者、用户或来宾账户。建议系统管理员在实现他们的安全策略时遵循这样的内置分组: 只有当他们有充分的理由时才定义具有不同许可模式的分组。

一些预定义组 (predefined group) 是同活动目录一起安装的。此外, 有一些显式组也可以用于有效的定义访问许可。

- 所有 (Everyone): 包括了所有的本地和远程用户, 包括 Guest; 这个组可以用来允许或拒绝所有用户的许可。
- 交互 (Interactive): 包括了所有本地登录的用户。
- 网络 (Network): 包括了所有通过网络登录的用户。
- 系统 (System): 操作系统。
- 创建者 (Creator Owner): 文件或资源的创建者或所有者。

7.5.3 审计

Windows 2000 在安全日志中记录了与安全相关的事件。在请求对对象的访问时, 需要记录的与安全相关的事件可以在对象的 SACL 中定义。日志文件中的表项是由 SRM 生成的。与安全相关的时间通常包括了有效的和无效的登录尝试、特权使用, 以及创建、删除或者打开资源 (文件)。通过事件查看器 (Event Viewer) 可以选择和显示需要记录的时间。

审计日志的最大大小可以设置。当日志到达最大大小的时候, 可以定义三种包裹选项来处理。

- 根据需要覆盖事件: 为了给新的表项腾出空间, 可以覆盖任何记录。
- 覆盖 [x] 天之前的事件: 在指定天数之前的事件可以被覆盖。
- 不覆盖事件: 不覆盖已有的事件; 在记录新的事件之前, 必须手工清除日志。

采用后两种选项时, 系统管理员必须设置系统以便在日志满了的时候, 系统可自动关闭。注册表中的 CrashOnAuditFail 表项是按如下方式设置的。

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Lsa
Name: CrashOnAuditFail
Type: REG_DWORD
Value: 1
```

为了达到橘皮书 C2 要求, 这个略带破坏性的设置是必须的。

7.5.4 小结

由于有大量支持安全管理的概念工具, 尤其是在应用层的安全管理策略上, 与 UNIX 相比, Windows 更靠近人一机尺上的用户端。

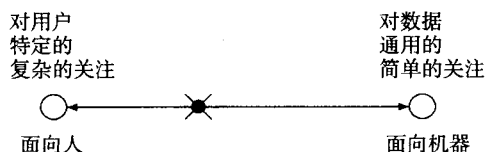


图 7-8 人一机尺上的 Windows 安全

7.6 深层阅读

针对特定操作系统的大多数安全手册没有超越安全系统的表象,并且专注于已有的安全特色和它们的管理。对于这种类型的参考,你必须检查最近的公布清单以及微软公司 <http://www.msdn.microsoft.com> 网站上的资料。*Brown(2000)* 深入讲解了 Windows 访问控制。

如果你对 UNIX 以及 Windows 上的操作系统安全实例研究感兴趣,你将发现大量深入研究 AS/400 安全的描述。AS/400 是 IBM 公司推出的运行于中型计算机上的操作系统。它是应用于金融领域的面向对象的操作系统,并且人们认为它相当安全。中间件层(CORBA)的访问安全是 *Lang and Schreiner(2002)* 的研究课题。

7.7 练习

练习 7.1 创建一个演讲课程容器。容器应该包括不同类型的对象:课程描述、演讲笔记、练习解答、学生项目。定义有关容器和对象的组和 DACL 以便:

- 课程的领导者有权访问所有资源;
- 参加课程的学生有读演讲笔记的权限,并且有读/写他们自己项目的权限;
- 所有学生有读课程描述的权限;
- 被任命为辅导员的学生具有对习题解答的读权限;
- 一个给出的习题解答对该课程的所有学生都是可见的。

练习 7.2 在 UNIX 和 Windows 2000 中,访问权限是为用户和组定义的。为了便于更好的安全管理,用户被放到不同的组中。当用户拥有的权限少于用户所在组的权限时,两个操作系统会如何决定访问权限呢?如何拒绝给予用户那些已经给予用户所在组的访问权限?

练习 7.3 讨论“中间层控制”是如何在 Windows 2000 中使用的?

练习 7.4 UNIX 的 UID 可以由管理员设置。Windows 创建随机的 SID。试讨论如下设计原理:就从管理员处获取控制并拥有由系统定义的随机描述符。

练习 7.5 默认的 Windows 账户不能被删除。试讨论设计决策的设计原理。

练习 7.6 应用软件可以强制执行它自己的安全控制。因而,开发者必须实现这些控制,在部署软件的时候,开发者并不知道这些参数(比如用户或组)将如何被实例化。试验证 Windows 2000 是怎样在安全策略的开发和部署之间提供接口的。

练习 7.7 在工作站全部都是单独管理的环境下,你将如何使用默认账户来管理计算机?

练习 7.8 试阐释在配置和审计安全操作系统中的主要问题(500 ~ 1000 字)。

第 8 章 Bell-LaPadula 模型

你的安全策略是什么？哪一条规则决定谁可以访问你的数据？为了制定一个安全策略，你必须描述策略控制的实体，并且陈述构成策略的规则。可以通过非正式的自然语言文档实现。在实践中，这些文档含糊不清、矛盾而且冗长。为了避免这些问题，你将更倾向于建立一个自己的正式的安全策略。安全模型正是在做这样的事情。

安全模型在高保险度安全系统设计和评估中具有突出的地位。其重要性已经在 1972 年的 Anderson 报告中体现出来了。设计过程始于系统应执行的策略的正式规范，即安全模型和系统自身的高级别规范。在该高级别规范中加入更多的细节，就可以获得一系列低级别规范。此时就得证明，高级别的规范能够实现预期的安全策略。对于高保险度，需要正式的证明。此外还得证明低级别的规范符合我们的安全策略，不过在这两个级别中，正式的证明变得越来越难了。

本章作为演示这种方法的案例进行研究。Bell-LaPadula 模型被研制来为分类数据捕获多级安全策略。我们会描述这个模型，并以 Multics 操作系统为例来说明，在分析某个专为执行这些安全策略的系统时，如何使用这个模型。下一章将介绍另外一些重要的安全模型。

目标

- 演示如何使安全策略形式化。
- 介绍 Bell-LaPadula 模型，并讨论其范围和局限性。
- 说明在分析一个安全系统时如何运用正式模型。
- 介绍计算机安全历史上几个重要的里程碑。

8.1 状态机模型

状态机(自动机)是针对计算机系统许多方面创建模型的一种流行工具，我们假设本书读者已经熟悉了这方面的内容。毋庸置疑，状态机也是一些重要的安全模型的基础。状态机模型的主要特点是状态(state)以及发生在离散时间点上的状态变化的概念。状态就是研究的系统在某个时刻的一种表现，它能准确捕捉与我们的问题相关的那些系统方面。可能的状态迁移(state transition)可以通过状态迁移函数来描述，该函数根据当前状态和输入来定义下一个状态，也可能产生出一个输出。

状态机的简单实例就是一个灯开关。它有两个状态，开和关，以及一个输入，按钮，这个按钮可以使系统从开状态转换到关状态，从关状态转换到开状态。一个更高级的实例就是自动售票机。它的状态需要记录售票请求及其需付金额。其输入是售票请求和钱币，输出便是票和所找换的零钱。计算机科学领域的一个例子就是微处理器。机器的状态由登记的内容决定，而它的输入便是机器指令。

如果我们想用状态机模型来讨论系统的某个特殊性质，如安全，首先必须确定达到该性质的所有状态，然后必须检查所有的状态迁移是否都保持(preserve)这种性质。如果是这样，并且系统是从满足这种性质的起始状态(initial state)开始运行的，那么可以通过归纳证明这个性质始终成立。

8.2 Bell-LaPadula 模型

Bell-LaPadula (BLP) 模型可能是安全模型中最著名的模型, 它由 Bell 和 LaPadula 在第一次联合设计安全的多用户操作系统时提出。如果这些系统是用来处理不同层次的机密资料的, 它们就必须执行多级安全 (MLS), 这在 4.7.3 节中已进行了详细描述。用户根据自己的许可应该只获得自己有权得到的信息。

BLP 是描述访问控制的机密性问题的状态机模型 (Bell and LaPadula, 1996)。访问许可通过访问控制矩阵和安全级别来定义, 安全策略防止信息从高安全级别流向低安全级别。BLP 只考虑了主体在查看或改变一个对象时发生的信息流动。

8.2.1 状态集

我们用第 4 章介绍的符号来描述 BLP 模型:

- 主体集合 S 。
- 对象集合 O 。
- 访问操作集合 $A = \{execute, read, append, write\}$, 直接借鉴了 4.3.2 节的访问权限。
- 具有偏序 \leq 的安全级别集合 L 。

我们想用系统状态来检查系统安全, 所以模型的状态集必须包括当前所有的许可和当前所有主体访问对象的实例。

我们可以用一个表来记录在特定的时间点哪个主体连接到哪个对象。表中的行索引是主体, 列索引是对象, 表中的一条记录显示了当前主体对对象执行的访问操作。在数学符号中, 这样的表对应于一个元组 (s, o, a) 的集合, 显示主体 s 当前正在对象 a 上执行操作 o 。元组是集合 $S \times O \times A$ (S, O, A 集的笛卡儿乘积) 的元素, 所以一个表对应于集合 $\mathcal{P}(S \times O \times A)$ 的幂的一个元素。它习惯用符号 b 来表示当前在 BLP 模型中进行访问操作的表, 而用 B 来表示这些表的集合。

目前, 获得访问许可的矩阵记作 $M = (M_{so})_{s \in S, o \in O}$ 。我们用 \mathcal{M} 来表示所有获得访问许可的矩阵的集合。

BLP 模型通过三个函数分别决定主体和对象的安全级别。

- $f_s: S \rightarrow L$ 给出了每个主体可以拥有的最高安全级别。
- $f_c: S \rightarrow L$ 给出了每个主体当前的安全级别。
- $f_o: O \rightarrow L$ 给出了所有对象的安全级别。

一个主体的当前级别不能高于它的最高级别, 因此 $f_c \leq f_s$, 读作“ f_s 控制 f_c ”, 很快我们就会看到引入 f_c 的原因。最高安全级别有时候称为主体的许可 (clearance), 其他文献只用许可来表示用户的安全级别。

为方便起见, 我们将三元组 (f_s, f_c, f_o) 记做 f , 用 $F \subset L^S \times L^S \times L^O$ 来表示所有可能的安全级别分配的集合。所有这些均导致了一个相当复杂的状态集 $\mathcal{B} \times \mathcal{M} \times F$ 。一个独立的状态由 (b, M, f) 这个三元组决定。定义状态集是 BLP 模型中的主要问题。我们不需要描述输入、输出或状态迁移的精确结构来给出 BLP 的安全特性。

8.2.2 安全策略

BLP 将安全定义为状态特性。仅当主体的安全级别控制对象的安全级别时, 多级安全策略才允许主体读这个对象。这些多级安全策略也称为强制性安全策略 (mandatory security policy)。

1. 简单安全性

第一个显而易见的特性是简单安全性 (ss-property)。

如果对于每一个元素都有 $(s, o, a) \in b$, 访问操作 a 是读或写, 主体 s 的安全级别控制对象 o 的安全级别, 即 $f_o(o) \leq f_s(s)$, 那么状态 (b, M, f) 满足 *ss-property*。

在 *ss-property* 中主体扮演查看者。当某人请求获取机密文件时, 这个特性就获得了传统的无向上读安全策略(*no read-up security policy*)。不过, 我们现在处于一个计算机系统中, 在里面主体就是过程。这样, 主体就不会像人一样具有记忆, 但它们可以访问记忆对象。主体能够作为通道, 读一个记忆对象并且将信息传送到另一个记忆对象中。用这种方法, 可能会对数据进行不适当的解密(图 8-1)。比如说, 黑客可能会插入一个高级别的特洛伊木马, 读高级别的对象的内容并将它拷贝到低级别的对象。

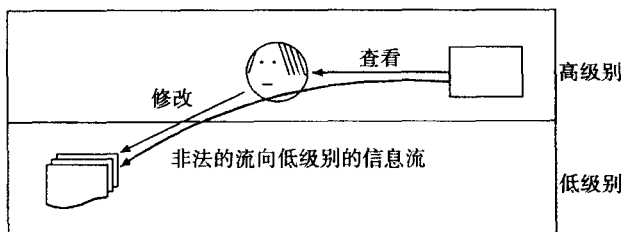


图 8-1 用主体作为通道对对象的解密

2. * -property

将用纸笔记录的安全策略转移到 IT 系统中会出现以前不曾有的问题。我们也需要一个策略来控制写访问。简单地在低级别阻止主体修改对象将会产生一个新的问题。这样的策略将会暗示高级别的主体不能给低级别主体发送信息。有两种方法可以避开这个限制:

- 临时地降低高级别主体的等级, 这是引入当前安全级别 f_c 的原因。
- 确定一组允许违背 * -property 的主体, 这些称为可信主体(*trusted subject*)。

第一种方法假设主体被降级时, 它会忘记它在较高安全等级上所知道的一切。当把主体看成人时, 这个观点看起来不合情理, 但 BLP 是模拟计算机的。在此主体(进程)没有自己的记忆, 它们所“知道”的唯一事情是允许它们查看的对象(文件)的内容。在这种情况下, 临时降级确实解决了问题。换句话说, f_s 指定了用户的许可, 用于允许以低于许可的级别注册, f_c 则表明了用户实际注册的级别。

BLP 包含了一个无向下写安全策略(*no write-down security policy*), 这个策略指向的是当前安全级别 f_c , 这个策略就叫做星特性(*star-property*, * -property)[⊖]。

如果对于每一个元素, 均有 $(s, o, a) \in b$, 访问操作 a 是添加或写, 主体 s 的当前级别被对象 o 的安全级别控制, 即 $f_c(s) \leq f_o(o)$, 那么状态 (b, M, f) 满足 * -property。

此外, 如果存在一个元素, 有 $(s, o, a) \in b$, 访问操作 a 是添加或写, 那么对于所有的对象 o' , $(s, o', a') \in b$, a' 是读或写, 我们必须有 $f_o(o') \leq f_o(o)$ 。

如图 8-1 所示, 往下流的非法信息已被 * -property 拦截。

当采用第二种方法时, * -property 只对不可信的主体有效。按定义, 一个可信主体可以违背安全策略。事实上, 为集中注意力, 你可以很好地用形容词“可信的”来识别那些会伤害你的系统组件。相反, 如果确信一个主体不会伤害你, 那么称它为值得信赖的(*trustworthy*)。

⊖ 该模型的第一个版本没有包括这个性质, 在为这个策略找到合适的名字前, * 一直被用作占位符。

3. 自主安全性

橘皮书使用术语自主访问控制 (discretionary access control, DAC) 来表示那些基于指定的用户和指定的对象的访问控制策略。拥有一项访问许可的主体可以将许可传递给其他主体。在 BLP 中, 这种策略通过一个访问控制矩阵表示, 并通过自主安全特性 (ds-property) 获得。

如果对于每一个元素, 均有 $(s, o, a) \in b$, 当有 $a \in M_{so}$, 那么状态 (b, M, f) 满足 ds-property。

8.2.3 基本安全定理

如果状态 (b, M, f) 满足 ss-, *-和 ds-property 策略, 那么称状态 (b, M, f) 是安全的。如果状态 $v_1 = (b_1, M_1, f_1)$ 和状态 $v_2 = (b_2, M_2, f_2)$ 都是安全的, 那么称从状态 v_1 到状态 v_2 的迁移是安全的。为了弄清楚需要进行什么样的检查才能确定一个新状态是安全的, 先考虑 ss-property。状态迁移保持 ss-property, 当且仅当:

- (1) 每个 $(s, o, a) \in b_2 \setminus b_1$ 满足关于 f_2 的 ss-property ($b_2 \setminus b_1$ 表示 b_2 和 b_1 的差集);
- (2) 如果 $(s, o, a) \in b_1$ 不满足关于 f_2 的 ss-property, 那么 $(s, o, a) \notin b_2$ 。

*-property 和 ds-property 特性的保持可以用类似的方法来描述。现在我们能够描述 BLP 模型的一个重要特性。

基本安全定理 如果系统中所有的状态迁移都是安全的, 并且系统的初始状态也是安全的, 那么不管输入情况如何, 其后的每一个状态也都是安全的。

该定理的形式化证明通过对输入序列的长度作归纳来进行。证明建立在这样的事实上, 即每个状态迁移保持安全, 且不引用特定的 BLP 安全属性。

经验教训

基本安全定理是状态机建模的产物, 而不是 BLP 模型中选用的特定安全特性的结果。

实际上, 基本安全定理减少了验证系统安全所需的努力。你可以分别检查每一个状态迁移以证明它保持了安全性, 并且需要确定一个安全的初始状态。只要以这种安全的初始状态启动系统, 它将会一直保持安全。

8.2.4 稳定性

McLean (1987) 提出了一种包含以下状态迁移的系统, 从而引发了一场关于 BLP 模型价值的激烈讨论:

- 将所有的主体降到最低的安全级别。
- 将所有的对象降到最低的安全级别。
- 在访问控制矩阵 M 的所有位置输入全部的访问权限。

按照 BLP 的定义, 这个迁移到达的状态是安全的。但这种状态能被视为是安全的吗? 既然 BLP 是这么说的, 那么 BLP 正确地获得了安全性了吗? 有两种意见:

- 反对 BLP (McLean): 直观地说, 如果一个系统能被带入到一种允许每个人读任何信息的状态, 那么这个系统是不安全的。因此 BLP 必须改进。
- 赞成 BLP (Bell): 如果用户要求这种状态迁移, 那么应当在安全模型中允许这种状态迁移。如果它不被要求, 那么就不应该去实现它。这不是 BLP 的问题, 而是正确获取安全需求的问题。

争论的根源是用于改变访问权限的状态转移。这种改变在 BLP 的一般性框架中当然是可能

的,但是模型的组织者实际上考虑的是访问权限固定的系统。安全级别和访问权限从不改变的这种特性称为稳定性(tranquility),从不改变访问权限的操作称为是稳定的(tranquil)。

8.2.5 BLP 的各个方面及其局限性

BLP 是一个非常重要的安全模型,它在安全操作系统的设计中起着重要的作用,而且几乎任何一种新的模型都要和 BLP 相比较。在这种情况下,分析 BLP 的一些特征是很有好处的。

(1)模型的描述能力:BLP 状态集描述了当前所有的访问操作和访问许可。

(2)安全策略是以安全级别和访问控制矩阵为基础的,在这些地方很容易引入其他结构。比如有这样一个情形,主体只被允许通过某些程序来访问对象,要建立这样一个访问控制模型, $S \times S \times O$ 访问控制结构更为合适。

(3)实际的安全特性:在 BLP 中,我们有 ss-property, *-property 和 ds-property。Biba 模型(参见 9.1 节)和 BLP 模型的主要区别在于安全特性。

(4)特定解决方案:比如, Multics 说明(参见 8.3 节)中的状态迁移。

BLP 用访问控制来定义安全性是它深受欢迎的一个主要原因。因此,用 BLP 来表示操作系统或数据库管理系统的行为是不太困难的。然而,尽管 BLP 是一个重要的安全模型,它却没有涵盖安全的所有方面。对它的批评有:

- 只涉及机密性,而没有设计完整性。
- 没有解决访问控制的管理问题。
- 包含隐蔽信道。

缺乏完整性策略是 BLP 的一个特征,而不是缺陷。在下一章你将会看到,对于一个安全模型来说,限制它的目标是非常合理的。BLP 没有针对修改访问权限的策略,事实上,BLP 最初打算用于安全级别不能改变的系統。

隐蔽信道(covert channel)就是一个不受安全机制控制的信息流(加拿大系统安全中心,1993)。如果低级别的主体可以看见高级别对象的名字,而只被拒绝访问对象的内容,那么对象的名字就是一个典型的隐蔽信道。在 BLP 中,你可以用访问控制机制本身构造一个隐蔽信道,信息可以像下面所示的那样从一个高安全级别流向一个低安全级别。

(1)低级别的主体在本级别上创建一个对象 dummy.obj。

(2)它的高级别同谋(一个特洛伊木马)提高或者不改变 dummy.obj 的安全级别。

(3)接着,低级别主体试图读 dummy.obj。该请求的成功或失败暴露了高级别主体的行为,1 位的信息从高级别流向了低级别。

告诉一个主体某个操作是不允许的,这本身就构成了信息流。这导致了数据库安全中有趣的解决方案(多实例化),一个对象在不同的安全级别上可能有不同的值,从而避免这类问题的发生(Denning et al., 1988; Lunt et al., 1990)。

经验教训

有时,仅仅隐藏对象的内容是不够的,可能也必须隐藏它们的存在。

8.3 BLP 的 Multics 阐述

Multics 操作系统(Multiplexed Information and Computing Service, 多元信息和计算服务)的研究目标是发展一个具有安全、可靠等性能的多用户操作系统(Organick, 1972; Bell and LaPadula, 1975)。在安全领域中的大部分研究,如 BLP,都是由 multics 工程触发的。在 Organick (1972, 第 4 章)中给出了对保护机制大体的看法。因为它的目标广泛性和它的安全要求, Mul-

tics 对工程成员来说变得太麻烦, 他们随后创建了更简单的操作系统, 即 UNIX。考虑到商业的成功因素, 这两个系统在平衡可用性和安全性方面表现突出。运行 Multics 的最后一个系统在 2000 年退出使用。

学习 Multics 给了我们一个了解安全模型(以 BLP 模型为例)如何用于安全操作系统设计的机会。作为访问控制的一个正式模型, BLP 非常适合获取操作系统的安全要求。事实上, 它就是为这个目的而开发的。BLP 中安全的引导定义使它相对来说更容易建立一个安全的系统。我们仅仅需要定义一个合适的状态迁移来保证安全。为了证明 Multics 是安全的, 我们必须找到一个和 BLP 一致的 Multics 的描述。我们大体上将跟从由 *Bell and LaPadula*(1975) 给出的陈述来显示 BLP 概念是怎样映射到 Multics 的。

8.3.1 Multics 中的主体和对象

Multics 中的主体就是进程。每个主体都有一个描述符段(descriptor segment), 这个段包含进程的相关信息, 以及进程当前访问对象的相关信息。对每一个对象, 在每个主体的描述符段中都有一个段描述符字(segment descriptor word, SDW)。图 8-2 给出了 SDW 的版式。SDW 包含了对对象的名字, 指向对象的指针, 以用于读、执行和写的指示器(indicator)标志。这些指示器指的是在 4.3.2 节中专门定义的访问属性。主体的安全级别存放在进程级别表(process level table)和当前级别表(current-level table)中。活动段表(active segment)跟踪记录所有的活动进程。只有活动进程才能访问对象。

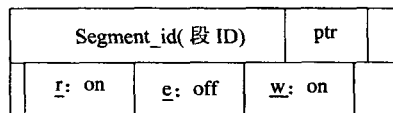


图 8-2 Multics SDW

Multics 中的对象是内存段落、I/O 设备等。对象在一个目录树中分等级组织。目录又是段。一个对象的信息, 就像它的安全级别或访问控制列表(ACL), 保存在对象的父级目录中。为了改变一个对象的访问控制参数, 创建或删除一个对象, 需要向父级目录写或追加访问控制权。

为了访问一个对象, 进程必须从根目录遍历目录树到达目标对象。如果路径中有任意一个目录不能被这个进程访问, 那么目标对象也不能被访问。换句话说, 在一个秘密的目录中, 一个非机密的用户不能读一个非机密的对象。因此, 将对象放入一个更高安全级别的目录中是毫无意义的, 并且我们总是要求一个对象的安全级别能够控制其父级目录的安全级别。这个属性叫做兼容性(compatibility)。在现代操作系统(如 UNIX)中你必须处理同样的问题。如果你希望你的文件可被其他用户访问, 也必须得到设置目录路径的访问权限。

BLP 的状态集组件的结合 Multics 现在我们拥有所有必需的信息来识别系统表和描述符段中的数据, 所有必要信息。

- 当前访问函数 b: 存储在活动进程描述符段中的 SDW 中; 在活动段表中可找到活动进程。
- 访问控制矩阵 M: 由 ACL 描述; 对于每个对象而言, ACL 均存储在对象的父级目录中; 每一个 ACL 表项都指定了一个进程和这个进程在那个对象上的访问权限。
- 级别函数 f: 主体的安全级别存储在专门的进程安全级别表、进程级别表和当前级别表中; 对象的安全级别存储在它的父级目录中。

8.3.2 转换 BLP 策略

4.3.2 节已经介绍了 Multics 对数据段和目录段的访问属性。我们也解释了这些访问属性如何与 BLP 模型中的访问控制权相对应。作为复习, 我们将重申数据段的访问属性, 作为提醒。

访问属性 访问权限

读 r
执行 e, r
读和写 w
写 a

BLP 安全特性现在改称为进程和数据段以及存储在 SDW 中的指示器的安全级别。比如，对 * - property 的阐释如下。

对于一个活动进程的描述符段中的任何 SDW，进程的当前级别为：

- 如果读或者执行指示器处于开启状态而写指示器处于关闭状态，那么就支配段级别。
- 如果读指示器处于关闭状态而写指示器处于开启状态，那么就被段级别支配。
- 如果读指示器处于开启状态并且写指示器也处于开启状态，那么就等价于段级别。

图 8-3 显示了怎样验证 * - property 的兼容性。当前进程的安全级别 L_c 保持在当前级别表中。描述符段基址寄存器(descriptor segment base register, DSBP)的内容指向当前进程的描述符段的头部。这个描述符段碰巧为一个对象包含了 SDW，在此访问控制权为只读。因此写指示器处于开启状态而读指示器处于关闭状态。对象的安全级别 L_o 是从它的父级目录推导出来的，并且与 L_s 相比较以检查 $L_s \geq L_o$ 成立。

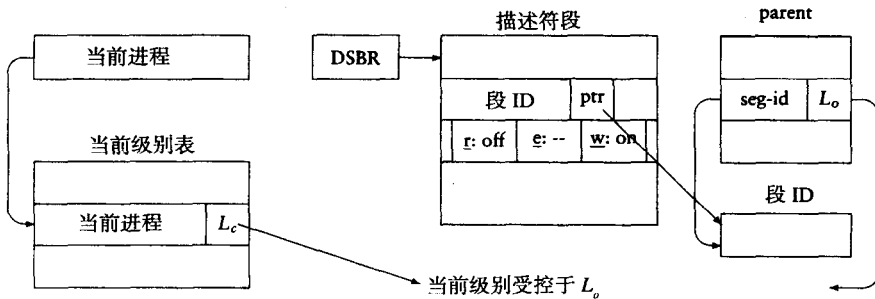


图 8-3 访问属性 Write Only 的 * - property

8.3.3 检查内核原语

最终，必须专门定义一套内核原语(kernel primitive)。这些内核原语在 Multics 内核的抽象模型中是状态的迁移，我们必须证明它们保持了 BLP 安全策略。然后，在基本安全定理成立的前提下，我们拥有 Multics 安全性的证据。当然，这不是一个完整的安全性的证明。你仍然必须说明内核原语的实现过程，以及最后它们在一个给定的硬件平台上的执行以符合它们的规范。

我们选择 get-read(获得阅读)来从细节上查看内核原语。get-read 原语将一个进程 ID 号和一个段 ID 号作为它自己的参数。操作系统必须核实：

- 段 ID 的 ACL 是否存储在段的父级目录中，是否列出了带有读许可的进程 ID。
- 进程 ID 的安全级别是否支配段 ID 的安全级别。
- 进程 ID 是否是一个可信任的主体或者进程 ID 的当前安全级别是否支配段 ID 的安全级别。

如果这三个条件都满足，那么就允许访问。如果不存在对段 ID 的 SDW，则有一个相应的 SDW 加到带有开启的读指示器的进程 ID 的描述符段中。如果在进程 ID 的描述符段中已经存在一个对段 ID 的 SDW，那么在这个 SDW 中的读指示器就会开启。如果这三个条件中的任意一个不满足，就不允许访问。

以下是更多的一些在 Multics 内核中建议执行的原语。

- release-read(释放读): 进程释放对象; 在相应的 SDW 中, 读标志关闭; 如果自此之后没有指示器开启, 这个 SDW 就从描述符段中移除。
- give-read(允许读): 一个进程授权对另一个进程的读访问权(DAC)。
- rescind-read(撤销读): 一个进程撤销准许的另一个进程的读许可。
- create-object(创建对象): 进程创建对象; 操作系统必须核实在对象目录段中的写访问是否被允许, 以及段的安全级别能够控制进程的安全级别。
- delete-object(删除对象): 删除对象时, 会产生和在 create-object(创建对象)时一样的核实。
- change-subject-current-security-level(改变主体当前安全级别): 操作系统必须核实在改变中没有产生安全蛮力; 该内核原语和原始的改变对象安全级别, 一样都不是为实现(稳定性)而设计的。

理想情况下, 要研发处理器以便使它们的指令集与操作系统的内核原语相吻合。相反的, 内核原语必须设计为能与已有处理器提供的支持相匹配的形式。

8.4 深层阅读

BLP 模型的最初报告最近在《Journal of Computer Security》(Bell and Lapadula, 1996)上重新发表了。McLean(1990)的著作中讨论了 BLP 模型中改变访问权限的策略框架的论题。该研究著作在保持 MLS 策略的同时扩展 BLP 模型的范围方面作出了贡献。

Denning(1982)在他的著作中综合论述了针对首个多用户操作系统的计算机安全研究的成果。Landwehr(1983)包含了保护技术的进一步调查。Multics 安全评论, 尤其是安全管理的复杂性评论以及对设计的访问的正确性、复杂性的评论可参见 Saltzer(1974)的著作。Multics 安全研究的经验讨论参见 Karger and Schell(2002)的著作。

8.5 练习

练习 8.1 用基本访问模式术语 alter(修改)和 observe(查看)描述 *-property。

练习 8.2 识别 BLP 模型中更深层次的隐蔽信道。

练习 8.3 写一篇短文表明你在 Bell 和 McLean 争论中的立场。

练习 8.4 BLP 没有规定改变访问权限的策略, 你会建议什么样的策略?

练习 8.5 改写针对 Multics 操作系统的 ss-property。

练习 8.6 详细说明必须为允许写(get-write)和释放写(release-write)内核原语而需要进行的检查。

练习 8.7 详细说明必须为创建对象(create-object)和删除对象(delete-object)内核原语而需要进行的检查。

练习 8.8 详细说明必须为改变主体当前安全级别(change-subject-current-security-level)内核原语而需要进行的检查。

第9章 安全模型

BLP 模型被设计用来捕获一种特定的安全策略。它是如此成功，以至于有一段时间人们把它视为通常意义上的“安全模型”。毋庸置疑该模型在此方面也存在着不足，但这并非是吹毛求疵。而是由于安全需求是取决于应用的，同时，现实生活中存在与作为多级安全起源的军事环境非常不同的应用。

本章将更广泛地探讨安全模型。我们将增加完整性策略的模型(Biba, Clark-Wilson)，增加与 BLP 静态环境假设形成鲜明对比的、动态改变访问权限的策略模型(Chinese Wall, 中国墙)。我们也将更广泛地使用“安全模型”这个词。例如，Clark-Wilson 模型并未给出一个简单的特定策略的形式，但它给出了一个可描述的框架，该框架可以作为一个蓝图使安全策略的宽泛类别正式化。我们还将进一步讨论来自理论观点的主要的有价值的模型，它们提供了一个证明某些有关访问控制基本事实的基础。

目标

- 介绍访问控制的模型化技术的广泛范围。
- 介绍商业安全策略的相关基本概念。
- 提供分析访问控制问题的理论基础。
- 了解安全中的一些决策问题本质上是不可判定的。

9.1 Biba 模型

考虑用完整性级别格(L, \leq)的元素标注主体和对象的完整性策略，该策略禁止“干净的”高级别实体被“脏的”低级别实体污染。在完整性的格中，信息只能向下流动。正如在 BLP 模型中一样，我们只关心自己的由访问操作直接引发的信息流。我们使用“干净的”和“脏的”来速记高完整性和低完整性。完整性级别的具体含义取决于给定的应用。

Biba 模型(1977)定义了这类完整性策略的格式。它类似于 BLP 的状态模型，我们将使用前一章引入的数学符号。赋予主体和对象的完整性级别由函数 $f_s: S \rightarrow L$ 和 $f_o: O \rightarrow L$ 给定。不像 BLP 模型，这里没有唯一的高级别的完整性策略。相反，存在不同的方式，有的甚至会产生相互不兼容的策略。

9.1.1 静态完整性级别

对应 BLP 的稳定性，我们可以规定完整性级别不变的策略，以下两个策略阻止干净的主体和对象被脏的信息污染。

简单完整性 如果主体 s 可以修改对象，那么 $f_s(s) \geq f_o(o)$ 。(无向上写)

完整性 * - property 如果主体 s 可以读(查看)对象 o ，那么仅当 $f_o(p) \leq f_o(o)$ 时， s 可以写访问另一个对象 p 。

这两个完整性特性是强制性 BLP 策略的对偶，也是主张完整性是机密性的对偶的基础。

9.1.2 动态完整性级别

如果一个实体已经接触了低级别的信息,下面两个完整性属性将自动调整(adjust)该实体的完整性级别。完整性级别 $\inf(f_s(s), f_o(o))$ 是 $f_s(s)$ 和 $f_o(o)$ 最好的低限,它在我们处理一个完整性级别的格时被很好地定义。

主体低水印性 主体 s 可以读(查看)任何完整性级别上的对象 o , 主体的新的完整性级别是 $\inf(f_s(s), f_o(o))$, 其中 $f_s(s)$ 和 $f_o(o)$ 是操作前的完整性级别。

对象低水印性 主体 s 可以修改任何完整性级别上的对象 o , 对象的新的完整性级别是 $\inf(f_s(s), f_o(o))$, 其中 $f_s(s)$ 和 $f_o(o)$ 是操作前的完整性级别。

这些是动态改变访问权限策略的例子。由于完整性级别只能降低,故存在所有的主体和对象最终终止于最低的完整性级别的危险。

9.1.3 调用的策略

可以扩展 Biba 模型以包含访问操作调用。主体可以调用(involve)另一个主体(如软件工具)来访问对象,这是向在中间层上表达的访问控制迈出的第一步。应当用什么样的策略来控制调用?如果我们想确保调用不会绕过强制性完整性策略(mandatory integrity policy),我们可以加入

调用性 主体 s_1 可以调用主体 s_2 , 仅当 $f_s(s_2) \leq f_s(s_1)$ 。

主体只允许在较低级别上调用工具,否则的话,一个脏的主体可使用一个干净的工具去访问,并且可能污染一个干净的对象。

另一种情况是,我们可能仅想因此目的而使用工具。脏的主体应该可以访问干净的对象,只要它使用一个干净的工具来访问(受控的调用),这个工具可以执行许多一致性检查,确保对象仍然保持干净。操作系统中的保护环(4.6.4节)完整性保护机制就属于此类。在这种情形下,我们可能不希望拥有较多特权的主体使用较少特权的工具,我们可以采用

环属性 主体 s_1 可以读任何完整性级别上的对象,它只能在 $f_o(o) \leq f_s(s)$ 时修改对象 o , 仅当 $f_s(s_1) \leq f_s(s_2)$ 时可以调用主体 s_2 。

很显然,最后两个特性是不一致的,我们必须根据应用来决定哪种特性更合适。

9.2 中国墙模型

Brewer and Nash(1989)提出的中国墙模型模拟了咨询公司的访问规则,分析员必须保证他们与不同客户的交易不会引起利益冲突。通俗一点说,冲突的产生是因为客户是同一个市场的直接竞争者或者是因为公司的所有权。分析员必须遵循以下安全策略:

一定不能有引起利益冲突的信息流。

Bell-LaPadula 模型的状态集需要一些小的调整来实现这个策略:

- 公司的集合用 C 表示。
- 分析员是主体,而 S 是主体的集合。对象是信息条目,每一对象均指向一个独立的公司,对象的集合用 O 表示。
- 有关同一公司的所有对象集中放在一个公司数据集(company dataset)中,函数 $y: O \rightarrow C$ 给出了每个对象的公司数据集。
- 利益冲突类(Conflict of interest)指出哪些公司在相互竞争,函数 $x: O \rightarrow P(C)$ 给出了每个

对象的利益冲突类, 即不应该知道该对象内容的公司集合。

- 对象 o 的安全标签 (security label) 是 $(x(o), y(o))$ 。
- 净化的信息 (Sanitized information) 被去除了敏感的细节, 不受访问限制的控制。一个被净化的对象的安全标签为 $(\phi, y(o))$ 。

利益冲突的产生不仅与当前访问的对象有关, 也与过去访问过的对象有关, 因此我们需要一个记录主体行为的数据结构。为此目的, 我们引入了一个布尔型 $S \times O$ 矩阵 N , 其中

$$N_{s,o} = \begin{cases} \text{真 (true)}, & \text{如果主体 } s \text{ 访问过对象 } o \\ \text{假 (false)}, & \text{如果主体 } s \text{ 从未访问过对象 } o \end{cases}$$

对所有的 $s \in S$ 和 $O \in O$ 设置 $N_{s,o} = \text{false}$, 将得到一个填充如下安全属性的初始状态。

第一个安全策略涉及直接信息流。我们希望阻止一个主体遭受利益冲突, 因此仅当对象请求属于以下情形时访问才被允许:

- 用户已经拥有的一个公司数据集。
- 一个完全不同的利益冲突类。

我们可以将它正式地表示成 *ss-property*。

主体 s 允许访问对象 o , 仅当对于所有 $N_{s,o'} = \text{true}$ 的对象 o' , 有 $y(o) = y(o')$ 或 $y(o) \notin x(o')$ 。

该属性本身没有完全实现规定的安全策略, 仍有可能发生间接信息流。接下来一起看看下面的例子 (图 9-1)。两个竞争者, A 公司和 B 公司, 在同一家银行开有账户。处理 A 公司与银行业务的分析员 A, 用 A 公司的敏感信息更新银行资产负债表; 处理 B 公司与银行业务的分析员 B, 现在可以访问到竞争者的经营信息。因此, 我们引入 **-property* 来控制写访问。

主体 s 允许对对象 o 进行写访问, 仅当 s 对 $y(o) \neq y(o')$ 和 $x(o') \neq \phi$ 的对象 o' 没有读访问权。

仅当没有其他的位于不同公司数据集, 且包含非净化信息的对象能被阅读时, 对一个对象的写访问才是允许的。在图 9-1 的例子中, 两个写操作将被 **-property* 属性阻止。*-property 属性阻止未净化的信息流出公司的数据集。

与访问权限通常假设为静态的 BLP 不同, 你现在看到的模型是在每次状态迁移中访问权限都要重新赋值。

9.3 Clark-Wilson 模型

Clark and Wilson (1987) 研究了商务应用的安全性需求。他们认为这些需求主要是关于 (数据) 完整性的, 即防止未经授权的数据修改、欺骗和错误。这是一个相当广泛的完整性定义。事实上, 作者甚至在其中包括了并发控制的问题, 该问题超出了我们的安全范围。完整性要求被分成两部分:

- 内部一致性 (Internal consistency): 涉及系统内部状态的属性, 可通过计算系统实施。
- 外部一致性 (External consistency): 涉及系统内部状态和真实世界的联系, 必须通过计算系统以外的手段来实施, 如审计。

实施完整性的一般机制是:

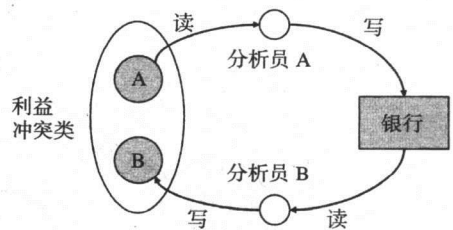


图 9-1 利益冲突类中的间接信息流

- 组织良好的事务(well-formed transaction): 数据项只能通过一组特定的程序来操作; 用户可以访问程序而不是数据项。

- 责任分离(separation of duties): 用户必须协同起来才能操作数据和穿透安全系统。

责任分离在一个安全系统的操作中反复出现。因而要求不同的人开发、测试、证明和运作一个系统是合理的。反过来, 也可能要求不同的人在操作中协同工作才能实现一个事务。

Clark-Wilson 模型用程序作为主体和对象(数据项)之间的中间控制层, 主体被授权执行某些程序, 数据项可通过特定的程序访问(图 9-2)。定义可以访问某个特定类型数据的程序集是软件工程的一种常用机制[见抽象数据类型(abstract data type)——Denning, 1982; 面向对象程序设计(object-oriented programming)], 这种机制可以被卓有成效地运用到构造安全的系统中。Clark 和 Wilson 写到“用程序而不是安全级别来标记主体和对象”, 这是对 BLP 影响力的一个有力证明。

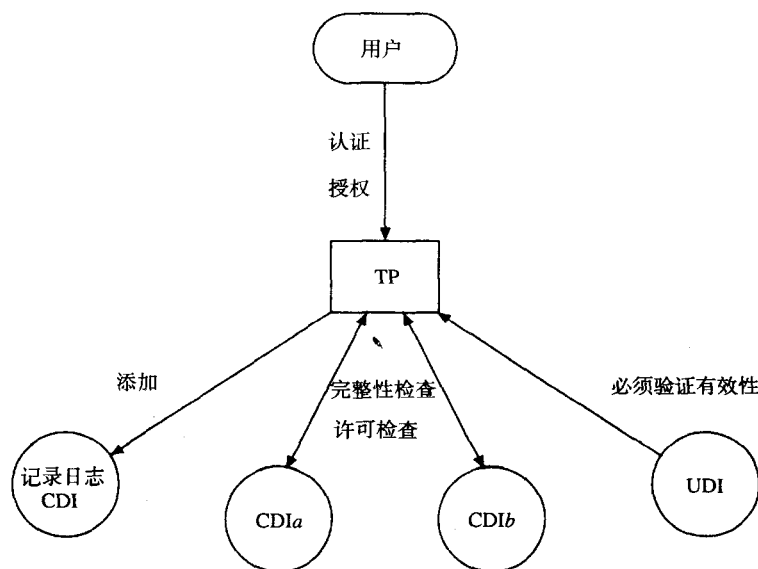


图 9-2 Clark-Wilson 模型中访问控制的基本原理

在 Clark-Wilson 模型中, 完整性意味着“被授权将一个程序应用到可以被该程序访问的一个数据项上”。Clark 和 Wilson 强调了军事安全与商业安全要求之间的区别, 观察表明机密性和完整性的相对重要性在这两个世界中是不同的, 但是也存在着有完整性要求的军事应用和有机密性要求的商业应用。对我们来说, 还有一个更相关的区别, Clark-Wilson 模型中的访问操作是执行复杂的特定应用操作的程序, 而 BLP 中的访问操作是简单的、通用的和适合操作系统的, 因此我们看到了通用操作系统(BLP)和面向应用的 IT 系统(Clark-Wilson)之间的区别。

总的说来, Clark-Wilson 模型考虑以下几点:

- (1) 主体必须被识别和认证。
- (2) 对象只能通过一组规定的程序进行操作。
- (3) 主体只能执行一组规定的程序。
- (4) 必须维护一个正确的审计日志。
- (5) 系统必须被证明能够正确工作。

在该模型的正式描述中, 安全策略所控制的数据项被称为约束数据项(CDI), 系统输入作为非约束数据项(UDI)。UDI 到 CDI 的转换是系统关键的一部分, 它不能光靠系统中的安全机制来

控制。CDI 只能通过变换程序 (TP) 来操作, 状态的完整性通过完整性验证程序 (IVP) 来检查。

安全属性是通过五条证明规则 (certification rule, CR) 定义的, 这些证明规则说明了为使安全策略符合应用需求而应该进行的检查。

CR1 IVP 必须确保当 IVP 运行时所有 CDI 处于有效状态 (CDI 上的完整性检查)。

CR2 TP 必须被证明是有效的, 即有效的 CDI 必须总是被变换成有效的 CDI。每个 TP 获准访问一组特定的 CDI。

CR3 访问规则必须满足任何责任分离要求。

CR4 所有 TP 必须写入一个只能添加的日志中。

CR5 任何一个接受 UDI 作为输入的 TP, 要么必须将 UDI 转换成 CDI, 要么必须拒绝该 UDI, 并且不进行任何变换。

四条实施规则 (enforcement rule, ER) 描述了应实施安全策略的计算机系统其内部的安全机制, 这些规则同 BLP 中的自主访问控制有些类似。

ER1 系统必须维护和保护 (CDIa, CDIb, ...) 列表, 该列表给出了 TP 获准访问的 CDI。

ER2 系统必须维护和保护 (TP1, TP2...) 列表, 该列表规定了用户可以执行的 TP。

ER3 系统必须认证每一个请示执行 TP 的用户。

ER4 只有可以授予 TP 访问规则的主体才能修改列表中相应的表项, 这个主体必须不具有在该 TP 上的执行权限。

最后请注意, Clark-Wilson 模型实际上是设计安全策略的一个框架和指导方针 (“模型”), 而不是一个具体安全策略的模型。

9.4 Harrison-Ruzzo-Ullman 模型

BLP 模型没有就改变访问权限或创建、删除主体与对象的规定策略, Harrison-Ruzzo-Ullman (HRU) 模型定义了解决这些问题的授权系统 (authorization system)。为描述 HRU 模型, 我们需要:

- 一个主体集合 S 。
- 一个对象集合 O 。
- 一个访问权限集合 R 。
- 一个访问矩阵 $M = (M_{so})_{s \in S, o \in O}$; 元素 M_{so} 是 R 的子集, 说明了主体 s 对对象 o 拥有的权限。

有六种原语操作 (primitive operation) 用于处理主体集、对象集和访问矩阵:

- **enter** r into M_{so}
- **delete** r from M_{so}
- **create subject** s
- **delete subject** s
- **create object** o
- **delete object** o

HRU 模型中的命令有下格式:

command $c(x_1, \dots, x_k)$

if r_1 in M_{s_1, o_1} **and**

if r_2 in M_{s_2, o_2} **and**

\vdots

```

if  $r_m$  in  $M_{s_m, o_m}$ 
then
     $op_1$ 
     $op_2$ 
     $\vdots$ 
     $op_n$ 
end

```

下标 s_1, \dots, s_m 和 o_1, \dots, o_m 是参数表 (x_1, \dots, x_k) 中出现的主体和对象。条件列表检查特定的访问权限是否出现, 条件列表可以为空。如果所有的条件成立, 那么基本的操作序列将被执行。每个命令至少包含一个操作, 比如命令

```

command create_file( $s, f$ )
    create  $f$ 
    enter  $o$  into  $M_{s, f}$ 
    enter  $r$  into  $M_{s, f}$ 
    enter  $w$  into  $M_{s, f}$ 
end

```

是由主体 s 用来创建一个新的文件 f , 因而 s 是文件的属主(访问权限 o), 并且对文件有读和写的许可权限(访问权限 r 和 w)。文件 f 的属主 s 用以下命令将读访问授予另一个主体 p :

```

command grant_read( $s, p, f$ )
    if  $o$  in  $M_{s, f}$ 
    then enter  $r$  in  $M_{p, f}$ 
end

```

访问矩阵描述了系统的状态。命令的结果作为访问矩阵的变化被记录下来。习惯上用 M' 表示修改后的访问控制矩阵。HRU 模型能够获得调整访问权限分配的安全策略。为了验证一个系统符合这种策略, 必须检查不存在授予不希望的访问权限的情况。

称一个状态(即访问矩阵 M)泄漏(leak)了权限 r , 即是指如果存在一个命令 c 将权限 r 加到了访问矩阵中一个原先不包含 r 的位置上。更正式的表述是, 存在 s 和 o , 使得 $r \notin M_{s, o}$, 但是 $r \in M'_{s, o}$ 。

称一个状态(即访问矩阵 M)关于权限 r 是安全的, 即是指如果没有一个命令序列能够将 M 变换到泄露 r 的状态。

因此, 在 HRU 模型中验证安全策略的符合性归结为验证安全属性(参见 9.6 节)。然而, 现在你会发现你处于一个相当尴尬的情形。

定理 给定一个访问矩阵 M 和一个权限 r , 则 M 关于权限 r 的安全性是一个不可判定的问题(Harrison, RuZZD and Ullman, 1976)。

因此, 你不能一般性地解决安全问题, 而必须通过约束模型来获得更好的成功机会。比如, 你可以只允许单操作系统(mono-operational system), 该系统中每个命令只包含单一操作。

定理 给定一个单操作的授权系统、一个访问矩阵 M 和一个权限 r , 则 M 关于权限 r 的安全性是一个可判定的问题(Harrison, Ruzzo and Ullman, 1976)。

限制授权系统的大小是使得安全问题易于处理的另一种方法。

定理 如果主体的数量是有限的, 那么自由授权系统的安全问题是可判定的(Lipton and Snyder, 1978)。

这些关于安全问题可判定性的结论揭示了第三条设计原则的大致内容(2.4.3节)。如果设计的复杂系统只有用复杂的模型来描述, 那么要找到安全证明相当困难。在最坏情况下(不可判定性), 不存在一个验证所有问题实例安全性的通用算法。如果需要可验证的安全特性, 最好限制安全模型的复杂性。这样的模型可能无法描述所有希望获得的安全特性, 但是你可以获得有效的验证“安全性”的方法。因此, 设计能够用简单模型充分描述的简单系统是明智之举。如果系统和模型间存在太大的差距, 模型中的安全证明就没有多大意义了。

经验教训

安全模型的表达能力(包括安全属性和它能够描述的系统)越强, 验证安全属性通常也越困难。

9.5 信息流模型

在 BLP 模型中, 信息可以通过隐蔽信道从一个高安全级别流到一个低安全级别。信息流模型(information flow model)考虑了所有形式的信息流, 而不光是由 BLP 模拟的通过访问操作的直接信息流。口语化的表述为, 如果能够通过观察 y 获得更多关于 x 的信息, 状态迁移即会引发从对象 x 到对象 y 的一个信息流。如果已知 x , 那就没有信息能从 x 流出。可以区分以下两种情况:

- 显式信息流: 在赋值“ $y := x;$ ”后查看 y , 揭示 x 值。
- 隐式信息流: 在条件语句“if $x = 0$ then $y := 1;$ ”后查看 y , 可以获得有关 x 的一些信息——即使赋值“ $y := 1;$ ”没有被执行。比如, 如果 $y = 2$, 可知 $x \neq 0$ 。

9.5.1 熵和平均值

精确和定量的信息流定义可以用信息理论的观点给出。从一个观察对象得到的信息量可以用我们正在观察的对象(变量)的熵来定义。设 $\{x_1, \dots, x_n\}$ 为变量 x 的可能取值, 设 $p(x_i)$ 为 x 取 x_i 值的概率, $1 \leq i \leq n$ 。则 x 的熵 $H(x)$ 定义为

$$H(x) = - \sum_{i=1}^n p(x_i) \log_2 p(x_i)$$

例如, 设 x 取 0 到 $2^w - 1$ 之间的所有值的概率相等, 则

$$H(x) = - \sum_{i=1}^{2^w} \frac{1}{2^w} \log_2 \left(\frac{1}{2^w} \right) = w$$

即, 如果所有长度为 w 的单词具有等同的可能性, 则一个长度为 w 的二进制单词携带 w 比特的信息。

x 到 y 的信息流用给定 y 的值后的 x 的信息量平均值(条件熵)的变化来度量。令 $\{x_1, \dots, x_n\}$ 和 $\{y_1, \dots, y_m\}$ 分别是变量 x 和 y 为概率 $p(x_i)$ 和 $q(y_j)$ 的可能取值, 设 $p(x_i, y_j)$ 为 x 和 y 取 x_i 和 y_j 值的联合概率, 设 $p(x_i | y_j)$ 为 y 取 y_j 值时 x 取 x_i 值的条件概率, 则给定 y 的值后的 x 的熵 $H_y(x)$ 定义为

$$H_y(x) = - \sum_{i=1}^n \sum_{j=1}^m p(x_i, y_j) \log_2 p(x_i | y_j)$$

重写 $p(x_i, y_j)$ 为 $p(x_i | y_j)q(y_j)$ 可以得到

$$H_y(x) = - \sum_{j=1}^m q(y_j) \sum_{i=1}^n p(x_i | y_j) \log_2 p(x_i | y_j)$$

作为一个例子, 可考虑上面的赋值“if $x = 0$ then $y := 1;$ ”, 设 x 和 y 为二进制变量, y 的初始

值设为0, x 的两个值很可能相等, 有

$$p(0|0) = p(1|1) = 0, p(1|0) = p(0|1) = 1, \text{ 因此 } H_y(x) = 0$$

事实上, 在执行赋值后观察 y , 可以知道 x 的确切值, x 中的所有信息流入了 y 。如果 x 为0,

1, 2 的概率相等, 将得到 $q(0) = \frac{2}{3}, q(1) = \frac{2}{3}$,

$$p(0|0) = p(1|1) = p(2|1) = 0, p(1|0) = p(2|0) = \frac{1}{2}, p(0|1) = 1, \text{ 且 } H_y(x) = \frac{2}{3}$$

9.5.2 基于格的模型

信息流模型的组成部分是:

- 一个安全标签的格(L, \leq)。
- 一组加了标签的对象集。
- 安全策略: 仅当 $c_1 \leq c_2$ 时允许从标签为 c_1 的对象到标签为 c_2 的对象的信息流; 任何违反该规则的信息流均非法。

如果没有非法的信息流, 系统就被称为是安全的。这种模型的好处是它覆盖了所有类型的信息流。其缺点是设计安全的系统变得更加困难。比如, 已经证明, 要检查在信息流模型中给定系统是否安全, 是一个不可判定的问题。

此外, 我们必须区分信息流策略的静态和动态实施。前者, 系统(程序)被看成是一个静态对象。后者认为系统处在执行中。我们可能发现有些信息流在理论上是可能的(因此应当在静态分析中检测), 却永远不会在执行过程中发生。因此静态分析趋向生成限制性过强的系统。

不干扰(noninterference)模型是信息流模型的替代方案, 它们提供了一个不同的形式来描述主体所知道的系统状态。如果 s_1 的行为对 s_2 的系统视图没有影响, 那么主体 s_1 不干扰主体 s_2 。目前, 信息流和不干扰模型只是研究领域, 而不是设计安全系统的实用方法论基础。

9.6 执行监控器

前面的两节显示了某些安全问题是不可判定的, 不存在一个应对这类问题所有情形的一般算法。现在, 我们的理论研究在走一条不同的途径, 从目前使用的典型访问控制机制和描述这些机制可能实施的策略的特征入手。毕竟一条策略只有能被合理有效地实施, 它才是实际有用的。我们来考虑安全策略的三个级别(Schneider, 2000)

- 访问控制策略定义了能够在对象上作业的操作主角的限制。
- 信息流策略能够限制主角通过观察系统行为而推断出的对象的信息(参见9.5节)。
- 可用性策略限制主角拒绝其他人使用资源。

目前, 访问控制机制通常部署在防火墙、操作系统、类似 CORBA 的中间件架构或 Web 服务中, 这些机制观察(监控)着系统的运行, 并且在某步执行操作被给定的安全策略禁止时干预该操作。执行监控器(Execution Monitoring, EM)这个词由 Schneider(2000)引入, 用于指这种机制的实施——监控目标系统的执行步骤并在违反安全策略的事件即将发生时终止目标的执行。

执行监控器有两个重要的限制。第一, 它们不包括目标系统的模型, 所以不能预测观察的执行继续下去的可能结果, 而且它们也不能修改目标系统。例如, 编译器和定理证明程序的工作是通过分析目标系统的静态表现, 可以推导出目标所有可能执行的信息, 因此这些方法不属 EM 机制。第二, EM 机制在一个目标执行前不能修改它。因此内联引用监控器和面向对象系统中的反射器不属于 EM 一类。

9.6.1 执行属性

目标系统的执行是一系列的步骤, 这些步骤的准确性质依赖于实际的目标。典型实例要数内存访问操作和文件访问操作。在我们的一般性讨论中, 令 ψ 表示所有的有限和无限的步骤序列的集合, Σs 表示代表目标系统 S 的执行的序列。安全策略 p 定义为执行集合上的一个谓词。如果 $p(\Sigma s)$ 为真则目标 S 满足安全策略 p 。HRU 模型的安全属性是安全策略的提取 (BLP, Biba 和中国墙策略也是如此)。

令 Σ 表示一个执行的集合, 一个可由执行监控器实施的安全策略 p 必须用一个如下形式的谓词指定

$$p(\Sigma): (\forall \sigma \in \Sigma: \hat{p}(\sigma))$$

其中 \hat{p} 是一个单独执行上的谓词, 这个观察提供了一个指向线性时间并发程序验证文献的连接 (Alpern and Schneider, 1985)。其中, 如果一个元素的成员关系由该元素自身独立决定而不是由集合中其他成员决定, 则执行的一个集合 $\Gamma \subset \Psi$ 称为一个属性。因此, 一个安全策略必须是一个具有 EM 中实施机制的属性。

然而, 并非所有的安全策略都是属性, 某些安全策略不能被定义为单独执行上的一个谓词。例如: 信息流策略要求低级别用户不能区分高级别用户主动参与的执行和未主动参与的其他执行。

此外, 也不是所有的属性都是 EM 可实施的, EM 中的实施机制对一个执行做出判定时不能看到将来的情形。考虑这样一个执行 σ , 它遵从安全策略但是具有一个不遵从安全策略的前缀 σ' , 口语化的表述是, 该执行会经历一个“非安全”的状态, 但最终会被允许。作为一个简单的例子, 可以参看: 要求为每一个“打开文件”命令匹配“关闭文件”命令的策略。一个执行监控器必须禁止一个非安全的前缀并停止后面安全的执行。就这样的策略而言, EM 成为了一种保守的方法, 它会阻止一些没有必要阻止的执行。

9.6.2 安全性和活动性

在执行属性中有两个应用广泛的类特别重要:

- 安全属性: 不会发生没有糟糕的情形。(HRU 模型中访问矩阵的“安全”属性实际上适合此处的描述。)
- 活动属性: 某些好的情形最终会发生。

在安全性和执行监控器可实施的策略类型之间存在密切的关系。我们将通过描述它们的补充特征来正式定义安全属性。在此定义中, 序列 $\sigma \in \Psi$ 的前 i 步骤用 $\sigma[..i]$ 来表示。如果对于每一个有限或无限的执行, 下式均成立, 则属性 Γ 被称为安全属性 (Lamport, 1985)。

$$\sigma \notin \Gamma \Rightarrow \exists i (\forall \tau \in \Psi: \sigma[..i] \tau \notin \Gamma)$$

如果一个执行 σ 是不安全的, 那么一定存在某个点 i , 在该点之后不再可能返回到安全的后续执行中。

如果一个安全策略的执行集合不是一个安全属性, 则存在一个可通过后续步骤扩展到安全执行的不安全执行。正如我们上面讨论的, 这些属性(策略)没有来自 EM 的实施机制, 因此, 如果策略不是安全属性则它不是 EM 可实施的。抛开其他方法, 执行监控器实施的是安全属性的安全策略。然而, 并不是所有的安全属性都具有 EM 实施机制的。

让我们回到开始时的安全策略的三个级别, 来总结一下我们的发现:

- 访问控制策略定义安全性; 以尝试不能被接受的操作而终止的部分执行将被禁止。

- 信息流策略未定义属于属性的执行集合；因此，信息流不能是安全属性，也不可能被 EM 实施。
- 可用性策略未定义安全性；任何部分的执行均可被扩展，因此主角最终将获得资源的访问权。

涉及最大等待时间(MWT; *Gliger, 1984*)的可用性策略是安全属性。一旦某个执行等待超过了 MWT, 则任何扩展自然也就违反了有效性策略。

9.7 深层阅读

安全模型方面的研究调查参见 *Landwehr(1983)* 和 *McLean(1994)* 的著作。*Clark and Wilson(1987)* 的原创论文是值得大力推荐的读物。*Karger(1991)* 的著作描述了 Clark-Wilson 模型的使用性能的实现。*T. M. P. Lee(1991)* 的著作中提出 Biba 模型的一种轻度扩展提供了多种强制性用于实现 Clark-Wilson 模型的完整性控制。HRU 模型和信息流模型的可判定性特性的详细介绍, 包括定义、证明和更多的理论, 请参见 *Denning(1982)* 的著作。非干扰模型, 请参考 *Goguen and Meseguer* 论文。*Bieber et al.(2000)* 和 *Schellhorn et al.(2000)* 的著作描述了智能卡安全评估中安全模型的应用。

9.8 练习

练习 9.1 Biba 模型可以捕获很多完整性策略, 举出以下策略适合的应用领域的例子:

- 具有静态完整性标签的策略。
- 具有动态可变完整性标签的策略。
- 环属性。

练习 9.2 你能够用 Bell-LaPadula 和 Biba 同时建立机密性和完整性模型吗? 你能在两个策略中使用相同的安全性标签吗?

练习 9.3 你能将中国墙模型放入 Bell-LaPadula 框架吗?

练习 9.4 中国墙模型中的 * -property 应当只指当前的读访问还是指任何过去的读访问?

练习 9.5 给出一个描述 Clark-Wilson 实施规则的规范模型。

练习 9.6 令 x 为一个 4 比特变量, 可在 0 到 15 中间等概率地取值。给定条件“if $x > 7$ then $y := 1$,”和初值“ $y = 0$ ”, 计算条件熵 $H_y(x)$ 。

练习 9.7 为 30 年后解密的文档设计一个安全模型。

练习 9.8 在一个控制对病历和处方访问的医疗信息系统中:

- 医生可以读写病历和处方。
- 护士只能读写处方, 但是不应当知道病历的内容。

你如何在格模型中得到这个策略, 阻止信息从病历流到处方? 依你看, 哪一种安全模型最适合这种策略?

第10章 安全评估

安全系统的用户需要其正在使用的产品能提供足够安全性的某种保证。他们可以：

- (1) 依靠制造商/服务提供商的保证。
- (2) 自行测试安全系统。
- (3) 或者依靠独立机构的公正评测(评估)。

用户必须是安全领域的专家,才能采用第2种方法。但是大多数的用户并不是安全领域的专家,因此某种安全评估是信任一种安全产品的唯一选择。本章将要探究安全评估和讨论目前的评估方案是否可以收益。

目标

- 了解所有评估过程都必须解决的基本问题。
- 提出一种比较评估标准的方法。
- 给出主要评估标准的概况。
- 评价一些已评估产品和系统的优点。

10.1 引言

可信计算机安全评估标准(TCSEC, 橘皮书; 美国国防部, 1985)是第一个获得广泛认可的安全评估标准。之后,人们又开发出了一些其他的标准,以解决橘皮书已知的不足。早先萌发的统一各种不同标准的愿望,最终导致了今天的共同标准(CCIB, 2004a)。安全评估标准开发过程中的里程碑有:

- 信息技术安全评估标准(ITSEC; 欧洲共同体委员会, 1991)。
- 加拿大可信计算机产品评估标准(CTCPEC; 加拿大系统安全中心, 1993)。
- 联邦标准(美国国家技术标准局和国家安全部, 1992)。

我们将通过提出下列问题来构造我们对评估标准的讨论。

1. 评估对象

评估标准一方面涉及产品,即已出售的部件,它将被用于多种应用且需要满足一般的安全要求,例如操作系统;另一方面涉及系统,即满足给定应用特定需求的各种产品的集合体。在第一种情况下,我们必须找到一个可一致接受的一般需求集。橘皮书的安全分类(security class)以及“联邦和共同标准”的保护配置文件(protection profile)都是为了达到上述目的。在第二种情况下,需求获取和分析已经成了每个独立评估的一部分。ITSEC就适合对系统的评估。

产品和系统间的区别突出了安全评估基本面上的进退两难的局面:用户不是安全专家但又有特定的安全要求。依据共同标准对已出售产品的评估抓住了典型的需求,对非专家来说这是个非常实用的决策标准,但是产品可能不会满足实际的安全需求。对定制系统的评估可解决我们感知到的需求,但是我们要使非专家用户确信我们已经正确获得了安全需求。在这一阶段如果能提供更进一步的帮助,我们才能开始跨过这一界线,即一般公共目的的安全评估和指导特定用户的安全顾问的任务之间的界线。

2. 评估的目标

橘皮书给出了如下概念之间的区别:

- 评估(*evaluation*): 评定一个产品是否具有它所声明的安全属性。
- 证明(*certification*): 评定一个(已评估的)产品是否适合给定的应用场合。
- 鉴定(*accreditation*): 确定一个(已证明的)产品是否可以在给定场合里应用。

这些是橘皮书的术语。当然其他的评估标准会使用不同的术语或者对同样的术语有不同的用法。对各种具体活动的命名, 不及它们各自应用目标上的基本区别重要。

3. 评估的方法

评估的可信性与评估所使用的方法密切相关。下面这两种情形是一个评估方法应必须避免的:

(1) 已评估过的产品后来被发现含有严重的缺陷。

(2) 不同评估方法对同一产品的评价产生分歧。因此, 可重复性(*repeatability*)和可再现性(*reproducibility*)常常应该包含在评估方法必须满足的要求之中。

安全评估可以是面向产品的, 也可以是面向过程的。面向产品(检测)的方法用来检查和测试产品。这种方法比面向过程的方法能告诉我们更多关于产品的信息, 但是不同的评估可以给出不同的结果。这是个可信性问题吗?

面向过程(审计)的方法检查文档和产品开发的过程。这种方法代价小, 而且容易达到可重复的结果, 但是结果本身并不是很有价值。欧洲信息技术安全评估手册(*European Information Technology Security Evaluation Manual*; 欧洲共同体委员会)第1版就是一个可重复性以绝对性优势压倒内容的一个典型实例。这也是个可信性问题吗?

4. 评估过程的组织框架

安全评估应该就安全产品各种属性得出独立的、被公众认可的定论。一个独立的评估机构可以是一个政府部门(这种方式最早出现于美国), 也可以是一个合适的被公认的私有企业(如在欧洲; 联合王国 *ITSEC* 方案, 1991)。在这两种方式中, 政府支持评估过程并且颁布证书。被公认的评估机构必须自行颁布证书(如德国), 我们可以想象什么情况下评估专家约定俗成的经验证据可以代替正式的鉴定。

如果所有的评估都由某单一政府部门来操作的话, 那么创建一个更具结构化的管理机构来确保评估的一致性将几乎不必要。然而, 随着时间的推移, 依然存在着解释偏差(*interpretation drift*)[标准蠕变(*criteria creep*)]的危险。评估过程可能因为评估机构缺乏竞争和资源有限而变得缓慢。当有经验的评估专家因为私人机构的更高薪水而跳槽时, 也就存在着人员流动的问题。政府实体可能要收取评估费用, 不过在美国, 进行评估是免费的服务。

在一个私营评估机构环境中, 发证机构必须在不同机构中实施评估的一致性(可重复性、可再现性), 也可以像欧洲那样确认评估机构的评估结果。为了避免不同的解释, 标准的精确形式化变得很重要。评估付费化和经济压力会导致更快速的评估, 评估对发起人的资源需求也会变得更加可预期。另一方面, 人们必须警惕, 避免经济压力会导致不正确的评估结果。

更多的组织机构方开始关注评估的主办者、产品制造商、评估机构间的合同关系。还有, 对于开始评估、发放评估证书和已被评估过的产品改进后的再评估等应该有一个合适的过程。

5. 评估标准的结构

安全评估的目标是对一个产品/系统是否安全给出保证。安全和保证与如下方面有关系:

- 功能性(*functionality*): 系统的安全特性, 例如自主访问控制(DAC)、强制访问控制(MAC)、认证、审计。
- 效果(*effectiveness*): 对于给定的安全需求, 所用评估机制是否合适? 例如: 用户是否经足够强度的口令认证或者应用程序是否需要一个加密的挑战-回应(*challenge-response*)协议?

- 保证性(*assurance*): 评估的完全性。

橘皮书为典型的美国国防部需求定义了评估级别。因此,在评估级别的定义中要同时考虑上述三个方面。ITSEC 定义了一种灵活的可适应新安全需求的评估框架。因此,上面三个方面可独立解决。

6. 评估的花费和收益

除了支付评估费外,你还必须考虑间接花费,如花费在产生评估所需要的证据上的时间、评估员的培训时间、评估团体的联系时间。当考虑评估花费时,你可以再区分为已出售产品和专用系统两种不同的评估。在第一种情况下,评估的主办者能潜在地在大量用户中分散费用。在第二种情况下,评估主办者,或者单个客户不得不自己来承受所有的费用。

评估可以应政府制定的方针的要求进行,也可以受法律或某些工业标准的委托进行。从用户的体验角度看,评估还可以提高产品质量。

10.2 橘皮书

安全评估指南的工作在美国开始于 1967 年。这项工作产生了可信任计算机安全评估标准(橘皮书),即第一个评估安全产品(操作系统)的指导方针。虽然这些成果集中在“国家安全”部分,但橘皮书的编写者却是想创建一个更能被普遍应用的文档,它可以:

- 为用户提供一个可用于评定计算机安全系统可信程度的评判标准。
- 为计算机安全系统生产商提供指导。
- 当要求一个计算机安全系统时作为说明安全需求的基础。

安全评估会检查系统的安全关联部分,即可信计算基(TCB)(参见 5.1 节)。橘皮书中的访问控制策略很接近于 Bell-LaPadula 模型(参见 8.2 节),即自主访问控制和强制访问控制都以安全标记格为基础。引用监控器(reference monitor)验证了主体是否被授权访问他们需求的对象。

高保证性总是与正式的方法、简单的 TCB 以及结构化设计方法学相联系。Bell-LaPadula 模型对一个满足橘皮书安全策略的正规模型是一个合适的参考,但其他模型也可以用到 TCSEC 评估中。假设 TCB 具有更高的简单性有助于更综合的分析。因此,复杂系统一定会划入较低的评估类别中。

橘皮书中的评估类是设计来说明安全需求的典型模式的。因此,具体的安全特点需求和保证需求都是结合在这些评估类的定义中的。一个评估类描述的主题有:

- (1) 安全策略:用主体和对象来说明的强制的和自主的访问控制策略。
- (2) 标记对象:标签用于说明对象的安全敏感度。
- (3) 主体识别:单个主体必须能被识别和认证。
- (4) 问责性:必须保留与安全相关的审计日志。
- (5) 保证:主要涉及安全体系结构的操作保证;诸如设计方法、测试和配置管理的生命周期保证。

(6) 文档:安全系统的系统管理员和用户需要的安装以及使用安全产品的指南;评估者需要测试和设计文档。

- (7) 持续的保护:安全机制不可破坏。

橘皮书用这些标准来定义四个安全划分和七个安全级别。通过更严格的分析,较高安全级别的产品提供更安全的机制和较高保证性。这四个安全划分是:

- D—最小保护
- C—自主保护(“需要知道”)
- B—强制保护(基于“标签”)

• A—可验证的保护

橘皮书的安全级别定义是递增的。一个级别的所有需求自动地包括在更高级别的需求中。橘皮书是国家安全机构执行的独立于应用的安全评估的基础。

1. D—最小保护

这个级别，是适用于那些需要评估但达不到橘皮书最低级别要求的产品。

2. C1—自主安全保护

C1 级别的系统是协作用户在同一完整性级别处理数据的环境。建立在单个用户和/或组基础上的自主访问控制(DAC)，使用户可在一定控制级别上共享对对象访问。但用户必须证明自己的身份而且该身份必须被系统认可。

对于操作的保证而言，TCB 有自己的执行环境，而且必须有周期性验证 TCB 操作是否正确的特点。生命期的保证仅指对“明显缺陷”进行安全性测试。用户指南(橘皮书文档的一章)、可信操作手册(适用于系统管理员)、测试文档、设计文档等都要提供。总之，C1 系统适合应用于友好环境，并不刻意提供强安全性。

3. C2—受控的访问保护

C2 系统使用户对各自的行为负责。DAC 加强了对单个用户的控制粒度。现在还必须控制访问权限的传播。如果由 TCB 管理的对象含有以前的主体产生的信息(对象重用)，则主体不能访问该对象。正如 C2 级别中明确定义的：必须保留安全相关事件的审计踪迹。

测试和文档必须包含最新增加的安全特性，但保证仍然是适度的。测试仍然是为了寻找明显的漏洞。

一般说来，虽然 C2 系统有许多固有的弱点，但 C2 仍被认为对商业应用是最合理的安全级别(欧洲计算机制造联合会，1993)。大多数厂商都提供经 C2 评估过的操作系统或数据库管理系统的版本。有时，厂商也提供专门的应用程序，以帮助在 C2 兼容的配置条件下安装他们的系统(Park, 1995)。

4. B1—标记的安全保护

B 级别用于处理保密数据和实施强制的 Bel-LaPadula 策略。每个主体和对象都依据层次分类级别和非层次类别(参见 4.7.3 节)设置标签。标签的完整性必须得到保护。识别和认证用于确定主体的安全标签。

一旦保护建立在基于标签之上，就必须考虑当被标记过的对象输出到其他系统或打印机时会发生什么情况。解决方法应该依据输出通道的特性而定。通信和 I/O 信道可以是单级别，也可以是多级别。在多级别上的信道上，对象和标签一起输出。在单级别信道上，TCB 和授权用户一起指定输出信息的敏感级别。可被阅读的输出也必须加标签，比如，在敏感文档的每一页打印上密级。

为达到更高的保证性，要求一个非正规或正规的安全策略模型。测试和文档必须做得更加彻底，设计文档、源代码和目标代码必须进行分析，所有必须移除测试当中揭示的所有的缺陷。

然而，B1 级别对于 TCB 结构并不十分苛求。因此，像多级别安全的 Unix 系统或数据库管理系统等复杂软件系统已接受了 B1 验证。B1 级别更倾向于面对高度隔离环境的系统。

如下产品被鉴定为 B1 级别：System V/MLS(出自 AT&T)和许多来自像 Hewlett-Packard、DEC 和 Unisys 等销售商的其他操作系统，以及像 Trusted Oracle 7、INFORMIX-Online/Secure、Secure SQL(出自 Sybase)等数据库管理系统。

5. B2—结构化保护

B2 级别主要通过系统在系统设计时增加安全需求来提高安全保证性。强制性访问控制(MAC)也

包含对物理设备使用的控制。一旦用户的安全等级发生变化则必须通知他们。对于登录和初始认证必须有可信路径。

安全策略的正规模型和系统的描述的顶级规范(Descriptive Top Level Specification, DTLS)都是必需的。模块化是系统体系结构的重要设计特点。TCB 必须提供不同的地址空间来隔离不同的进程。硬件支持(比如分段)可支持内存管理。必须实现隐蔽信道分析(covert channel analysis), 并且必须记录那些可能会创建一个隐蔽信道的事件。安全测试应该证实: TCB 可在一定程度上抵抗穿透。来自可信信息系统的可信 XENIX 操作系统被鉴定为 B2 级。

6. B3—安全域

B3 系统可以高强度抵抗穿透(highly resistant to penetration)。B3 级别中的许多新成分都要与安全管理有关, 也必须支持安全管理员操作。审计机制监视着正在发生的或累积的与安全相关的事件, 并对可疑情形自动发布警告信息。必须建立系统失败后可信恢复机制(trusted recovery)。最小化 TCB 的复杂性和排除与安全无关的模块是大多数系统工程的任务。令人信服的观点必须在安全策略的正式模型和非正式的详细实验说明书(DTLS)之间达成一致。

出自 Wang Government Services 的 XTS-300(和 XTS-200)多种版本被鉴定为 B3 级。XTS-300 是一款运行在其 x86 硬件基础上的多级别安全操作系统(STOP)。

7. A1—可验证的设计

A1 级别功能相当于 B3 级, 它通过正规方法的使用达到了最高的保证性。策略和系统的正式规范和一致性证明都在很高程度保证性上显示了 TCB 是正确实现了的。A1 级别的要求如下:

- 安全策略的正规模型。
- 系统要具有正规顶级规范(Formal Top Level Specification, FTLS), 包括 TCB 的抽象定义。
- 模型和 FTLS 之间的一致性证明(正式的, 若有可能的话)。
- TCB 的实现非正式地证明了和 FTLS 的一致性。
- 隐蔽信道的正式分析(定时通道的非正式分析); 隐蔽信道的持续存在必须经过认定, 且必须限制带宽。

另外, 更严格的配置管理和分布控制(站点安全确认测试)要保证装在客户站点上的版本和(评估过的)主版本一样。

典型的 A1 级别的产品是网络组件, 比如 MLS LAN(来自 Boeing)和 Gemini 可信网络处理器。SCOMP 操作系统也被评估为 A1 级别。编写橘皮书标准时, 曾考虑定义比 A1 级别更高的安全级别, 这些级别主要是在系统体系结构、测试、形式描述和验证, 以及可信设计环境等方面增加更多的需求。如果评估复杂软件产品的难度降低了安全保证级别, 那么就没有必要进一步在这个方向开展工作。

10.3 虹系列

橘皮书是 NSA 和 NCSC^①出版的安全需求、安全管理和安全评估方面文档集的一部分。该系列中每一个文档均以其封面的颜色而为人们熟知。由于该系列有很多, 因此被合在一起称为虹系列。橘皮书中介绍的概念和术语对应于《可信数据库管理系统解释》(Trusted Database Management System Interpretation; Lavender/紫皮书; NCSC, 1991)和《可信网络解释》(Trusted Network Interpretation; 红皮书; NCSC, 1987)书中的特定方面。这些标准最初是为评估处理政府(军事)应用中的分级数据系统而开发的。

① 美国安全部和美国国家计算机安全中心。

10.4 信息技术安全评估标准

协调的欧洲信息技术安全评估标准(欧洲通信委员会, 1992)是由荷兰、英国、法国和德国在定义国家安全评估标准时的结果。第一个草案公布于 1990 年, 信息技术安全评估标准(ITSEC)正式作为推荐版本被欧盟委员会批准的日期是 1995 年 4 月 7 号。作为欧式文档, ITSEC 有多种语言版本, 这增加了统一解释标准的难度。

ITSEC 从橘皮书各种版本的经验教训中学到了东西, 取得了必然的进步。橘皮书被认为太严格太僵化了, ITSEC 致力于提供一个安全评估的框架, 以便新的安全需要提出后可以很容易地加入。在 ITSEC 中, 功能性和保证性之间的联系被打破了。应用于安全产品的标准也被用于安全系统。评估对象(Target of Evaluation, TOE)一词引入到 ITSEC 中。

评估的发起人决定了操作性需求和可能的威胁。TOE 的安全目标(security objective)进一步取决于法律和其他规则, 这些形成了所需的安全功能和评估级别。安全对象(security target)定义了评估相关的 TOE 的所有方面。它描述了评估对象的安全功能, 也可能是面对的威胁、对象和所用安全机制的细节。TOE 的安全功能可以独立地定义, 也可引用预定义的功能类(functionality class)定义。

E0 到 E6 七个评估级别表示了安全功能执行的正确性的保证级别。E0 代表不充分的保证。对于每一个评估级别, 该标准都列举了发起人提交给评估者的条目。评估者要保证这些条目已被提供, 并注意内容和表述的任何需求都已得到满足; 保证条目被清楚地提供, 或支持所要求的证据的产生结果。建议发起人/开发者与评估者之间紧密合作。

通过区分功能性和保证性需求, 以及考虑整体安全系统的评估, 欧洲安全评估标准能解决红皮书、可信数据库解释中不能解决的问题。由 ITSEC 提供的灵活性有时是优点, 有时却是缺点。记住在 2.4.3 节强调的安全评估基本方面的进退两难的局面: 不是安全专家, 用户如何判定一个给定的安全对象对他们是否是正确的?

10.5 联邦标准

评估标准链上的下一环节是美国的联邦标准(美国国家技术标准局和国家安全部, 1992)。他们采用了下一个必然的步骤, 在评估级别的定义上提供了更多的指导, 但仍然保留一定的灵活性。联邦标准坚持对产品的评估, 坚持评估级别的定义中的功能性和保证性之间的联系, 并独立于产品的保护配置文件来尽力克服橘皮书中严格的僵化结构。保护配置文件包括下面五部分:

- 描述的元素(descriptive element): 保护配置文件的“名字”, 包括要解决的信息保护问题的描述。
- 基本原理(rationale): 保护配置文件的基本判断, 包括威胁、环境、使用假设, 要解决的信息保护问题更详细的描述, 和遵从该保护配置文件的产品支持的安全策略指南。
- 功能需求(functional requirement): 建立产品提供的保护界线, 这样在边界内可以统计预期的威胁。
- 开发保证需求(development assurance requirement): 从初始设计到实现的所有阶段, 包括开发过程、开发环境、操作支持、开发根据。
- 评估保证需求(evaluation assurance requirement): 说明评估的类型和深度。

10.6 共同标准

安全评估在商业上的运用很吸引人, 评估证书应该被广泛地接受。在这个方向上, 第一步是

就评估标准的共同集合达成一致。因此，负责国家安全的各种评估组织走到一起，形成了共同标准编写委员会(CCEB)，产生了共同标准(CCIB, 1004a)，作为联合现有的和将继续出现的评估标准的一种努力，这些现有标准有 TCSEC、ITSEC、CTCPEC 和联邦标准。1999 年，共同标准(CC)还成为国际标准 ISO 15048。CCEB 也被 CC 执行委员会(CCIB)接替。

共同标准融合了各种以前的标准的思想。(这种融合的令人遗憾的一面是，读者必须面对大量的文档)。CC 是为适用于产品和系统的安全评估而开发，通用的术语评估对象(TOE)再次被使用。CC 放弃了 ITSEC 采用的对功能类和保证级别的严格分离，在使用保护配置文件和预定义安全级别上追随了联邦标准。安全对象(ST)表示了一个特定 TOE 的安全需求，如通过一个保护配置文件的引用。ST 是任何安全评估的基本要素。评估保证级别定义了一个评估当中什么是必须做的。

10.6.1 保护配置文件

为了指导用户，保护框架(PP)中集中了安全目标、原理、威胁、威胁环境和应用节点等信息。一个 PP 是一个适合特定用户需要的安全需求的(可重用)的集合。图 10-1 给出了 PP 的结构。用户团体应该开发自己的 PP 来捕获自己的典型安全需求。一些类似橘皮书中的 PP 被提供来作为示例，但任何人都可以写属于自己的 PP，并且有一个现成的用来添加和诊断新 PP 的过程。你可以找到用于以下的 PP：单级别和多级别操作系统、数据库管理系统、防火墙、可信平台模型、邮政计算器、自动现金售货机、电子棒、安全签名设备和智能卡安全的数个方面。

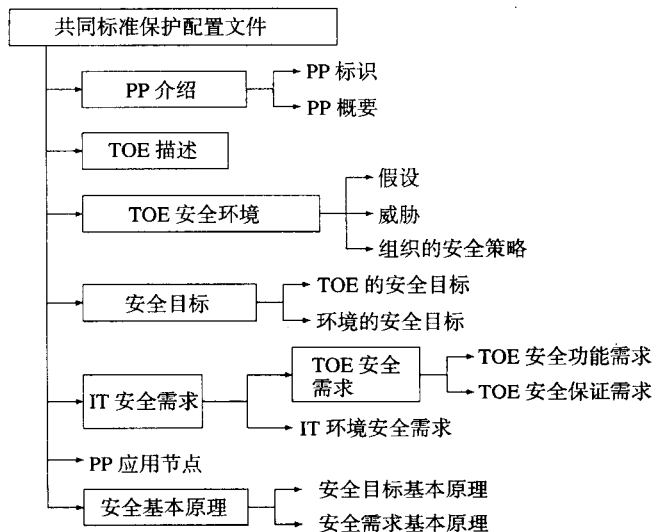


图 10-1 共同标准保护配置文件

10.6.2 评估保证级别(EAL)

EAL 定义了一个 TOE 的开发者和安全评估者的责任。有七个递增定义的 EAL。

EAL1—功能已测试 测试者收到 TOE，检查文档并进行一些测试来确认文档化的功能。评估不必要来自开发者的任何协助，评估的费用是最少的。

EAL2—结构已测试 开发者提供测试文档和来自漏洞分析的测试结果。评估者复查这些文档并重复其中一些测试。所需的来自开发者的努力较小，不需要有效的完整开发记录。

EAL3—方法已测试和验证 开发者使用配置管理器、文档安全管理器来开发,并提供高水平的设计文档和复查用测试覆盖文档。这个级别可用于已遵从好的开发实践并且不想对其实践执行深层改变的开发者。

EAL4—方法已设计、测试和复查 开发者为评估者提供低层次的设计和一个安全功能(TCB)源代码子集。还必须有一个安全的分发过程。评估者进行一个独立的脆弱性分析。通常EAL4是对已存的产品线经济可行的最高级别。

EAL5—半正式地已设计和测试 开发者提供一个正式的安全策略模型、一个半正式的高层次设计和功能规范及安全功能的完整源代码。必须进行一个隐蔽信道分析。评估者执行独立的渗透测试。对于这个级别的评估来说,如果TOE按达到EAL5保证的目的设计和开发将确实有显著帮助。超出开发过程本身成本的附加评估费用不会很高。

EAL6—半正式可验证设计和测试 源代码必须很好的结构化,访问控制(引用监视器)必须具有较低的复杂性。评估者必须进行更强的渗透测试,评估费用应该可预见地增长。

EAL7—正式可验证设计和测试 开发者提供正式的功能规范和高层次的设计。开发者必须展示或证明所有安全功能实现间的通信。对于正式分析而言,安全功能必须足够简单。这个级别典型地仅实现于这样的TOE:该TOE具有紧密聚集的安全功能并遵从广泛的正式分析。

10.6.3 评估方法

共同评估方法(CEM)定义了验证ST中保证性需求时必须遵从的所有步骤(CCIB, 2004b)。共同标准互认协定(CCRA)提供了对在其他国家中进行的评估的认可。CEM可用于保证级别EAL1到EAL4,只有这些保证级别被相互承认,更高的级别只在个别的国家被接受。在美国,共同标准评估和认证方案(CCEVS)是根据CC进行安全评估的国家程序。已有认证机构批准承担安全测试的实验室,提供技术指导和验证安全评估结果。

10.7 质量标准

以审计为基础的评估的最终一步,是在没有关于产品的任何参考的情况下评定产品是怎么开发的。这样的公司会成为“安全系统的认证制造商”。事实证明,这种方式在质量控制领域相当流行。诸如ISO 9000的标准会建议一个公司怎样摆正内部质量管理和外部质量保证的位置来保证产品的质量。一些厂家声称,对特定产品通过了ISO 9000质量认证取得的销售比一般只有安全证书的特定产品要强多了,安全评估也要走向这条道路。

这样的提议对公司开发安全系统的吸引力显而易见。评估的开销很大程度上减少了。如果安全系统的开发者的这个提议上成功了,安全系统的用户还会遭遇安全问题吗?这是个不可预测的结论。毕竟,一份证书并不能保证系统不可能被攻破。因此,不得不根据每一种评估机制自身的优点,来决定已单独评估过的产品是否能比授权开发者的产品提供更多的安全保证。

10.8 成果是否得到充分利用

在有些国家,公共部门的客户会要求根据CC进行安全评估。主要的操作系统和数据库管理系统供应商提供已经过评估的产品。然而,除政府部门外,人们对经评估产品的热情并不高。此外,也有例外,在某些市场中CC被大多数供应商追捧。在本书写作时,智能卡方面就是一个例子。早在十年前,即可指出PC安全软件的供应商。

安全评估被批评为一种由政府驱使的代价昂贵的过程。上面的观点参见欧洲计算机制造商联合会(ECMA, 1993)公布的报告。这个报告意识到了橘皮书C2级别在建立操作系统安全基线

中扮演的角色,但坚持要求成本、生产率和安全的三者平衡,其中的任何两项都会严重影响第三项。ECMA 警告不要把 IT 安全仅看成技术问题,并注意到了当前投资的一种失衡,即把过多努力放在安全评估上,却忽略了安全系统有效管理的重要性。该报告建议研究质量标准,如 ISO 9000,是作为安全评估的另一选择。

在他们对当前评估过程的批评中,评估成本(占开发成本的 10% ~ 40%)和到评估完成的时间当然是人们关心的方面。被提到的更多问题还有:

- 标准的解释和标准的推广上的不明确性,即对标准过时的解释的变化。
- 对已评估产品新版本的重新评估成本。
- 评估过程的保密性。

最后,你必须注意到证书只适用于产品的特定版本和特定配置。在一个实际安装中,使用的很可能是不同的配置,也可能是不同的版本。因此严格上说,证书并不能提供直接的安全保证。因此,曾有过一些开发评估方法学的尝试。这些方法使得以节俭的成本和较少的工作量再评估先前评估过的产品成为可能。虹系列中的 RAMP 方案就是一个这样的例子。

10.9 深层阅读

MacKenzie and Pottinger(1997)讲述了安全评估和正规安全模型化的早期历史。Chokhani(1992)概括了橘皮书级别总体的安全评估的实际应用方面。Schaefer(2004)的著作叙述了促使橘皮书以及其他文献诞生的 IT 发展。经 A1 级别评估的 BLACKER 系统的简短描述可在 Weissman(1992)的著作中找到。为达到 A1 需求开发的但没有成为商业产品的原型在 Karger et al.(1990)的著作中述及。作为一个安全保证和多级别安全系统的隐蔽信道方面的例子可以参见 Kang, Moore and Moskowitz(1998)的著作。

CTCPEC 被誉为最简洁和最具可读性的评估标准。包含评估标准、辅助文档和已评估产品列表的 Web 站点有:

- 虹系列网址为 <http://www.radium.ncsc.mil/tpep/library/rainbow>。
- CC 网址为 <http://www.csrc.nist.gov/cc> 和 <http://www.commoncriteriaportal.org/>。

10.10 练习

练习 10.1 安全评估要处理移动对象的问题。当某产品的一个版本被评估时该产品的开发仍然不能停止。怎样才能保证评估证书不过时呢?可参考 RAMP 方案。

练习 10.2 安全产品必须紧随正在变化的对象。在一个正在使用中的产品的生命期内,外部的威胁环境可能会改变。怎样设计一个方案来使防病毒产品的评估能在威胁变化的环境中保证它们的证书继续更新?在你的方案中有哪些组件应该包括在对操作系统的评估中?

练习 10.3 有时人们会这样评价,评估的产品主要用作在被指控没有遵照已设的最好机制时的保护措施,而不是为了能提供更好的安全性。从提供附加值的安全评估方案中,你希望得到什么?

练习 10.4 评估标准用来帮助对安全无知的用户满足特定的安全需求。保护框架是不是能正确解决这个问题?

练习 10.5 ITSEC 覆盖了安全评估。咨询顾问针对安全问题为客户推荐解决方案。从何处可以分清顾问的工作和评估工作?评估是否优于请雇用顾问?

练习 10.6 为防火墙写一个 PP(保护配置文件)。

练习 10.7 考查封锁和监控隐蔽信道的两种选择。被封锁的隐蔽信道影响的系统的可用性如何?

第11章 密码学

从前,早在20世纪80年代,在研究多级化安全的潮流中,你可能听说过密码学(cryptography)对计算机安全没有作出什么贡献。计算机安全就主要与TCB(可信计算基)、引用监控器、自主和强制访问控制,安全模型和系统规范的正规验证。按照这种观点,密码所起的作用实际上是外围的。在安全操作系统当中,存储口令的单向函数仅仅是密码机制的一个明显的实例。

直到20世纪90年代中期,却走向了另一个极端,密码被看成了可解决所有计算机安全问题的神奇对策。安全操作系统由于太昂贵,使用太受限制、太远离用户的要求,最终像恐龙绝迹一样消失在人们的视线中。在功能强大的密码算法中,输出限制被视为需要克服的障碍,并以此来保障计算机安全。又一个十年后,人们对密码学的贡献作出了一个更明确的断定,即,密码学能够为计算机提供安全。密码学技术仍然是安全分布系统的一个必要的组成部分。

目标

- 了解在不同目的下密码学的不同应用。
- 介绍密码学的基本概念。
- 理解密码能够解决问题的类型,以及在使用密码时需要解决的问题的类型。
- 指出支持密码学所必需的计算机安全特征。

11.1 引言

在传统定义中,密码学(cryptography)是书写秘密内容的学科,密码分析学(cryptanalysis)是分析和破解加密器(cipher)的学科,这两个主题就形成了密码学。它们曾经是间谍和秘密部门的领域,由于这些原因,现在密码仍然笼罩着一层神秘的面纱。

现代密码学完全是一门数学学科。对于很好地理解密码学出色的观点所必需的数学背景的阐述已经超出了本书的范围。我们将尽力解释怎样把密码学用于计算机安全,并且指出计算机安全常常是加密能起作用的一个先决条件。

11.1.1 旧的范例

密码在通信安全中有它自己的根源。通信安全解决了在图11-1中描述的情况。两个实体A和B通过一个不安全的信道通信。对手是一个入侵者,它完全控制了信道,能够读它们的信息、删除信息和插入信息等。两个实体A和B是彼此信任的。它们想保护通信不受入侵者的破坏。密码使得它们在不安全的物理连接上建立了一条安全的逻辑信道。在这方面,密码与到目前为止讨论的计算机安全机制是有本质区别的。对于来自底层的危害,它们全部都是脆弱的。然而,对物理通信链路的访问却不能威胁密码的保护。

在分布式系统中,客户端和服务端之间的通信流对所谓的入侵者来说是一个新的攻击点。不安全通信链路引入的漏洞自然能够由通信安全服务和机制来解决。这些服务包括以下方面。

- 数据机密性:加密算法隐藏了消息的内容。

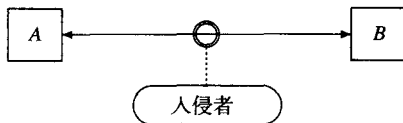


图11-1 通信安全

- 数据完整性：完整性检测功能提供了检测文档是否被改变的方法。
- 数据源认证：消息认证码或者数字签名算法提供了验证消息来源和它的完整性的方法。

数据源认证包括数据完整性。你不能声称已经验证了在传输过程中已经被更改了的消息的来源。另一方面，如果消息中包含了发送者的地址，当你在验证消息的完整性的同时，也就验证了消息的来源。在这种情况下，数据完整性和数据源认证是等同的概念。数据完整性的概念在其他应用中更有意义，比如，反病毒软件的文件保护。

谁是朋友，谁是敌人这种传统的观点也在计算机安全中有了它自己的位置，但它不再是驱动计算中密码应用的主要理由。遗憾的是，它仍然支配着公众对密码的理解。同样的观点也反映在用于密码协议的许多验证工具上，它们假设实体 A 和 B 的行为与协议规则完全一致，并且只考虑入侵者行为的影响。

11.1.2 新的范例

让我们关注新鲜的事务。在电子商务中，顾客进入了一个与商家有关的商业事务。两个参与者都不希望另一方欺骗，但是，纠纷却有可能发生，并且预先有一致的规则总比在发生纠纷时再采取特定的方式解决要好些。因而客户和商家都有理由运行一个协议，这个协议认为对方不是在任何情况下都是可靠的。现在的手是一个误操作的内部人员 (insider)，而不是一个入侵者，图 11-2 所示的第三个参与者不再是一个入侵者，而是一个可信的第三方 (TTP)，例如，一个仲裁者。在解决纠纷时，仲裁将要考虑不可否认服务 (nonrepudiation service) 生成的证据。

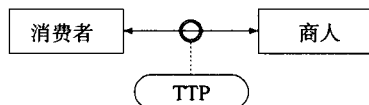


图 11-2 电子商务安全

许多国家都有法律，规定法律执行代理 (law enforcement agency, LEA) 在什么时候，怎样获得侦听许可证，以便责成电信服务提供者给他们接入到特定用户之间的通信的权利。现在，在图 11-3 中的第三方是一个电信操作员的客户，必须给他提供一个合法的侦听服务。在这种上下文中，密钥托管 (key escrow) 服务分发用于加密通信的密钥；密钥托管服务也是目前令人激动的研讨课题。

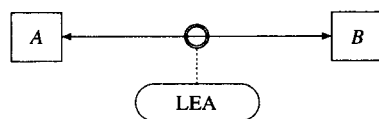


图 11-3 通信安全和法律实施

11.1.3 密钥

密码员特别喜爱采用锁作为他们的图标，以表示他们给公众提供安全服务。快速查看目前的具有安全性能 (security-enabled) 的 Web 浏览器或者 E-mail 产品的用户接口就能够证实这种观察。类推充满了危险，你不应该过分地信任它们，但是也有一些重要地概念从锁匠传承到了密码员。锁上门或者打开门上的锁，你需要一把钥匙。从强度上分，锁是不同的。有些锁很容易被撬开，而另一些锁是如此坚固，以致入侵者不得不采用暴力攻击破门而入，或者选择一条完全不同的路径，改为从房间的窗户进入房间。

密码算法使用密钥来保护数据。从现实方法的强度和范围来看又有各种变种，其中的一些使用简单的统计方法就可以破解，而另外一些远远超出了目前的数学分析和计算能力所能够达到的范围。蛮力攻击穷尽一切可能地搜索整个密钥空间，它给出了算法强度的上限。

现代密码学并不依赖于算法的保密。在密码变换中使用的密钥才是唯一需要保护的内容。在 19 世纪 Kerckhoffs 提出这个原则，在我们的新安全范型中非常适用，在这里必须支持大量具有竞争利益的用户团体。在这种环境下，形成事实上的标准和对公开算法的开放评估都是很自

然的过程,这会使得每一个参与者都有机会进行他们自己的安全评估,并且会使得新的参与者更容易加入。

因而,从最一般的字面意义上理解,密钥管理(key management)对密码机制的安全极为重要。你必须处理下面这些问题:

- 密钥在哪儿生成?
- 密钥怎样生成?
- 密钥在哪儿存储?
- 密钥怎样分发?
- 密钥实际上在什么地方使用?
- 密钥怎样撤销和替换?

到此,完成了一个循环,我们又回到计算机安全。密码密钥是存储在计算机系统敏感数据,计算机系统访问控制机制必须保护这些密钥。当访问控制失败时,密码保护面临着危险。在大多数目前实施的安全系统中,密码算法是最强的部分,且老谋深算的攻击者将寻找其他的软肋,而不是把他们的时间浪费在密码分析上。

经验教训

密码几乎从来都不是安全问题的解决方案。密码是一个转换机制,它通常把通信安全问题转化为密钥管理问题,而最终转化成了计算机安全问题。希望最后的问题比原来的问题更容易解决。总之,密码能够加强计算机安全,但它并不是计算机安全的替代品。

11.1.4 密码机制

密码机制是密码方案的最基本的构造模块。它们被用在密码协议中,且取决于好的密钥管理来提供有效的保护。最频繁应用于计算安全的密码机制是:

- 加密算法。
- 数字签名方案。
- 完整性检查函数(密码哈希函数,即 hash 函数)。

为了打破密码的传统,我们将以相反的顺序来介绍这些概念。首先,我们将解释一些关于模运算的基本知识,以作为后面描述内容的基础。

11.2 模运算

相当多的现代密码算法都建立在代数学原理的基础上。这些算法可以定义在先进的代数结构上,比如椭圆曲线或者伽罗瓦域(有限域)。然而,我们仍然有点停留在实际应用上,且在它们描述中仅使用整数。

令 m 是一个整数。在后面的描述中,我们将 m 称为模数(modulu)。然后在整数集合上定义了一个等价关系“ \equiv ”:

$a \equiv b \pmod{m}$, 当且仅当 $a - b = \lambda \times m$, 这里, λ 可取某些整数值。

我们则说“ a 和 b 模 m 同余”。可以检验“ \equiv ”确实是一个等价关系,它把整数集合划分成 m 等价类:

$$(a)_m = \{b \mid a \equiv b \pmod{m}\}, 0 \leq a < m$$

我们更习惯用 $a \pmod{m}$ 的形式来表示等价类,我们将遵循这个约定。你可以证明下列有用的性质:

- $(a \pmod{m}) + (b \pmod{m}) \equiv (a + b) \pmod{m}$

- $(a \bmod m) \times (b \bmod m) \equiv (a \times b) \bmod m$
- 对于每一个 $a \not\equiv 0 \bmod p$, p 是素数, 存在一个整数 a^{-1} , 使得 $a \times a^{-1} \equiv 1 \bmod p$ 。

对于一个模素数 p , 以 p 为模的乘法阶定义为:

令 p 是一个素数, a 是一个任意整数。 a 模 p 的乘法阶为满足下面关系的最小整数 n : $a^n \equiv 1 \bmod p$ 。

费尔马小定理表明, 任何一个以 p 为模的非零元素的乘法阶必定是 $p-1$ 的因子。

定理 对于每一个 $a \not\equiv 0 \bmod p$, p 是素数, 有 $a^{p-1} \equiv 1 \bmod p$ 。

这个事实被用于构造相当多的密码算法。这些算法的安全性常常是相关的, 在少数场合是等价的, 来自数论中的下述问题中的任何一个都是难解的:

- 离散对数问题 (Discrete Logarithm Problem, DLP): 给定一个作为模的素数 p , 基数 a 和值 y , 根据等式 $y = a^x \bmod p$, 求出 y 的离散对数, 即整数 x 。
- n 次方根的问题: 给定整数 m , n 和 a , 求出一个整数 b , 使它满足 $b = a^n \bmod m$ 。解 b 就叫做 a 的 n 次方根模 m 。
- 因式分解问题: 给定一个整数 n , 找出它的素数因子。

在参数的正确选择情况下, 这些问题是许多密码算法的合适的基础。然而, 并不是这些问题的所有实例都是难解的。很明显, 如果 p 或者 n 是比较小的整数, 那么这些问题就可以在合理的时间范围内采用穷举搜索来求解。目前, 二进制 512 位的整数已被认为是位数较短的整数了, 一般都推荐使用 1024 位整数, 当然, 如果你能够容忍由于算术运算占用更长的时间而引起性能的下降, 还可以使用更长的整数。长度不是你必须考虑的唯一方面。这些问题的难度也取决于 p 和 n 的结构(为了进一步追究这个课题, 你必须参照一些更专业化的数学方面的书籍)。

11.3 完整性检查功能

一个密码哈希函数 h , 将输入的任意比特长度的 x 映射到固定比特长度 n 的 $h(x)$ 输出。这就是众所周知的压缩性。哈希函数比提到的其他加密机制更快, 消耗资源更少; 计算方便的性质需要给定 x , 这样很容易计算 $h(x)$ 。

我们将讲解如何保护程序 x 不被篡改来说明如何利用完整性检查功能。这个策略运用在一些反病毒产品里。在一个没有病毒的环境下计算哈希值 $h(x)$, 并且把结果存储在不能修改它的位置, 例如, 存放在 CD-ROM 上。为了检查程序的状态, 重新计算哈希值, 把它与原来存储的值进行比较。哈希值的保护很重要。计算哈希值并不要求任何秘密的信息, 因此, 任何人都能对一个给定的文件计算有效的哈希值。

11.3.1 冲突和生日悖论

在我们的例子里, 重要的是不会找到冲突。如果有两个输入 x, x' , 且 $x \neq x'$, 使得 $h(x) = h(x')$ 成立, 我们就会有一个冲突。在这种情况下, 攻击者以哈希值不变的方式修改程序, 而对程序的改变不会被发觉。

通过暴力搜索找到一个冲突的概率取决于哈希值的比特长度。如果定义一个 n 比特哈希数 y , 则发现在 x 和 $h(x) = y$ 之前的预期数是 2^{n-1} 。然而, 如果仅仅是寻找任意冲突, 那么大约是 $2^{n/2}$ 组的输入很有可能有一组容易造成冲突。这个结果是基于生日悖论 (birthday paradox) 的。把 m 个球从 1 到 m 编号, 放进盒子里, 取出一个球, 列出它的编号并将它放回。重复这个实验。若

$m \rightarrow \infty$, 则在先前得出的数字之前预期数字趋近于 $\sqrt{\frac{1}{2}m\pi}$ 。

11.3.2 操作检测码

操作检测码(Manipulation Detection Code, MDC),也称为修改检测码或者消息完整码,常用于检查文档是否发生改变。不同的给定应用,在 MDC 上的要求也就不同。安全特性中来自一个哈希函数 h 的一个可能需求包括以下几种特性。

- 预映射抗拒性(preimage resistance)(单向性):给定 a 一个值 y ,为了找到满足 $h(x) = y$ 的 x ,通常这在计算上是不可行的。
- 第 2 次预映射抗拒性(second preimage resistance)(弱冲突抗拒性):给定输入 x 和 $h(x)$,求出另外一个 x' ,且 $x \neq x'$,使得 $h(x) = h(x')$,这在计算上是不可行的。
- 冲突抗拒性(collision resistance)(强冲突抗拒性):找到任何两个输入 x 和 x' ,使得 $h(x) = h(x')$ 成立,这在计算上是不可行的。

MDC 有两种形式(Menezes, van Oorschot and Vanstone, 1997)。

- 单向哈希函数(one-way hash function, OWHF)具有压缩性、易计算性、预映射抗拒性和第 2 次预映射抗拒性等性质。
- 冲突抗拒哈希函数(collision-resistant hash function, CRHF)具有压缩性、易计算性、第 2 次预映射抗拒性和冲突抗拒性等性质。

应用哈希函数计算的结果分别不同地称为:

- 哈希值(hash value)。
- 消息摘要(message digest)。
- 校验和(checksum)。

最后一个术语最容易引起混淆。在通信安全中,校验和是指错误校正码,典型的是循环冗余校验码(CRC)。CRC 是一个线性函数,所以容易产生冲突。另一方面,校验和也被用在反病毒软件的产品中,但是禁止使用 CRC 来计算,而必须使用密码哈希函数(MDC)来计算。

当明智地选择参数 p 和 g 时,函数 $f(x) := g^x \bmod p$ 是一个单向函数,这个函数称为离散求幂(discrete exponentiation)。为了反向离散求幂,你必须解决在 11.2 节介绍地 DLP(离散对数问题)。在这一章的后面可以看到离散求幂在密码方案的构造中是真正有用的原函数。然而,离散求幂并不是特别快的操作,当需要高速处理大量的数据时,必须转而采用其他的算法。

快速哈希函数倾向使用类似的设计模式来构造。哈希函数的核心是一个压缩函数 f ,它可处理固定长度的输入。一个任意长度的输入 x 被分割成给定块大小的块 x_1, \dots, x_m ,如果最后一个块没有达到固定长度,就使用填充的方法使它达到固定的长度。 x 的哈希值则可以通过反复的使用压缩函数获得。令 h_0 是一个(固定的)初始化值。计算

$$h_i = f(x_i \parallel h_{i-1}), \text{ 其中 } i = 1, \dots, m$$

且取 h_m 作为 x 的哈希值,如图 11-4 所示(符号 \parallel 表示联级)。

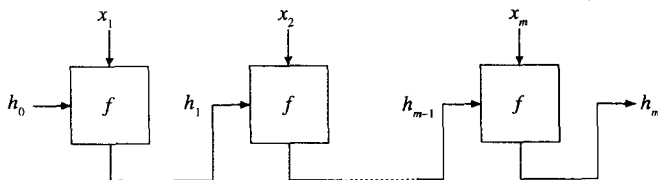


图 11-4 哈希函数构造

11.3.3 消息认证码

消息认证码(Message Authentication Codes, MAC)为消息的来源和完整性提供了保证(数据来源认证)。MAC从两个输入(即消息和秘密)的密码密钥计算得到。因此,MAC有时也称为密钥hash函数。正式地说,MAC是用密钥 k 作为参数化的hash函数 h_k 的族。这个族中的每一个成员都具有压缩和容易计算的特性,附加的抗计算性也必须满足。

对于任何固定的 k 值而言,给定一个组值 $(x_i, h_k(x_i))$,当攻击者不知道这个固定的 k 值时,对任何新的输入 x ,在计算上来说,要计算出 $h_k(x)$ 都是不可行的。

为了认证一个消息,接收者必须与发送者共享用于计算MAC的密码密钥,用来计算MAC。不知道密钥的第三方不可能确认MAC。使用下面的HMAC构造方法(Bellare, Canetti and Krawczyk, 1996; Krawczyk, Bellare and Canetti, 1997),可以从MDC算法 h 推导出MAC算法。为给定的密钥 k 和消息 x ,计算

$$\text{HMAC}(x) = h(k \parallel p_1 \parallel h(k \parallel p_2 \parallel x))$$

其中 p_1 和 p_2 是填充的位串,它把 k 扩展成在 h 中使用的压缩函数需要的完整的块长度。

11.3.4 安全哈希算法

我们将选择安全哈希算法(SHA-1)来说明哈希函数实际上^①是怎样设计的。这个算法被指定用于美国的数字签名标准(Digital Signature Standard, DSS)。其他的哈希函数是MD4(不具备强抗冲突性,参见Dobbertin, 1996),MD5(Internet协议中的标准选择,非强抗冲突性^②)和RIPEMD。SHA-1可处理任意多个512比特的数据块,生成一个160比特的哈希值。参数可以被作为整数解释也可以作为位串解释。我们在说明中省略掉了位串和整数之间的转换算法。

输入的末尾将用一个1来填充,然后是一个0串,以使得最后的输入块长度为448,且最后64比特字段指示在填充前输入数据的长度。初始化的值由五个32比特的值定义,以十六进制数表示。

A = 67452301

B = efcdab89

C = 98badcfe

D = 10325476

E = c3d2e1f0

SHA-1的压缩函数在一个80步的循环中处理512比特的输入,每20步改变一次内部函数和常数,最后生成一个160比特的输出。这些函数是:

$$f_t(X, Y, Z) = (X \wedge Y) \vee ((\neg X) \wedge Z) \quad \text{其中 } t=0, \dots, 19$$

$$f_t(X, Y, Z) = X \oplus Y \oplus Z \quad \text{其中 } t=20, \dots, 39$$

$$f_t(X, Y, Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z) \quad \text{其中 } t=40, \dots, 59$$

$$f_t(X, Y, Z) = X \oplus Y \oplus Z \quad \text{其中 } t=60, \dots, 79$$

操作符是位与、位或和异或,并被应用到32位字上。常数采用十六进制数表示,它们是:

$$K_t = 5a827999 \quad \text{其中 } t=0, \dots, 19$$

① 在写作本书时(2005年2月),已经报道了SHA-1上的冲突攻击。这类攻击限制了SHA-1被应用于其上的应用程序,并且极容易产生一个新的哈希算法标准。

② 在Eurocrypt 2005 M. Daum 和 S. Lucks 的著作的结尾部分介绍了对创建postscript对的通用攻击,该postscript对具有相同的MD5哈希值。

$$K_t = 6ed9eba1 \quad \text{其中 } t=20, \dots, 39$$

$$K_t = 8f1bbcdc \quad \text{其中 } t=40, \dots, 59$$

$$K_t = ca62c1d6 \quad \text{其中 } t=60, \dots, 79.$$

在压缩函数开始的时候, 五个 32 比特的变量 a, b, c, d 和 e 是用哈希函数的中间值来初始化的。对于第一个输入块, 则使用初始值 A, B, C, D 和 E 。512 比特的输入块被分成了 16 个 32 比特的字 m_i , 并且使用下面的算法, 将它们扩展成 80 个 32 比特的字 w_i :

$$w_i = m_i \quad \text{其中 } t=0, \dots, 15$$

$$w_i = (w_{i-3} \oplus w_{i-8} \oplus w_{i-14} \oplus w_{i-16}) \lll 1 \quad t=16, \dots, 79$$

其中, 符号 $\lll s$ 表示循环左移 s 位。现在的压缩函数执行下面的循环, 其中加法使用模 2^{32} 运算:

```
for  $t=0$  to 79 do
```

```
begin
```

```
temp =  $(a \lll 5) + f_i(b, c, d) + e + w_i + K_i$ ;
```

```
e = d;
```

```
c =  $b \lll 30$ ;
```

```
b = a;
```

```
a = temp;
```

```
end;
```

最后, 变量 a, b, c, d 和 e 的值被加入到前面的中间哈希值中。在处理下一个输入块的时候, 这个结果用作初始的哈希值。

11.4 数字签名

在图 11-1 中, 消息认证码可帮助参与者 A 和 B 检测在通信信道中由入侵者插入的欺骗消息。然而, 消息认证码并不生成任何证据, 供第三方用来判断 A 或者 B 是否发送了特定的消息。因此, 在图 11-2 的电子商务中, 顾客需要确保商家不能伪造订单, 且商家需要确保顾客必须认可他们所下的订单, 在这样的情况下, 消息认证码几乎是没有什么用处的, 因而数字签名 (digital signature) 就很有必要了。

数字签名方案由密钥生成算法、签名算法和验证算法组成。文档的数字签名是一个值, 它取决于文档内容, 并且仅有签名者知道的某个秘密, 即私有签名密钥。私有签名密钥把文档与一个实体联系在一起, 即公共验证密钥。验证算法通常使用文档或公共验证密钥作为输入, 但是在一些异常的情况下, 文档或者文档的一部分可以从签名中恢复出来, 且对于签名验证不一定要提供文档。图 11-5 给出了一个典型数字签名的示意图, 在该图中私人签名密钥应用于哈希文档。下述的可验证性表示了数字签名方案的特征:

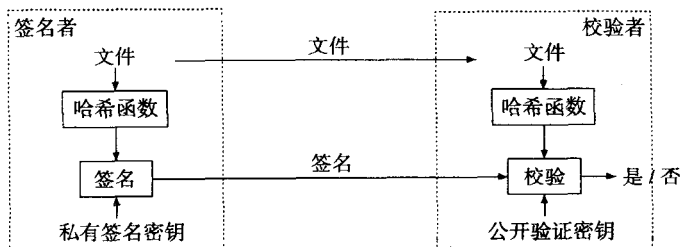


图 11-5 通用数字签名方案示意图

第三方可以在不知道签名者的私钥的情况下, 解决关于数字签名有效性的纠纷。

数字签名支持不可否认性。公钥加密(参见 11.5 节)是数字签名方案的自然来源。在这种方案中, 私有签名密钥和公共验证密钥之间的联系具有不可从验证密钥中推导出签名密钥的特性。不管基础数学技巧上是否有相似性, 都应该推断出数字签名和公钥加密算法之间清楚的区别。这两种方案从根本上满足了不同的技术目的。加密保护了消息的机密性, 因此必须是可逆的。数字签名提供了数据来源的认证和不可否认性。数字签名算法不必是可逆的。事实上, 可逆性会引起一些额外的安全担忧。

11.4.1 一次性签名

你不需要为构造一个签名方案而钻研数学。为了获得一次性签名, 你仅仅需要一个密码哈希函数 h 就可以了(Lamport, 1979)。为了对一份 n 位的文档签名, 随机选择 $2n$ 个值的 $x_{i,0}$ 和 $x_{i,1}$ 作为你的私钥, 且公布值(约束) $y_{i,0} = h(x_{i,0})$ 和 $y_{i,1} = h(x_{i,1})$, $1 \leq i \leq n$, 作为你的公钥。对文档 m 的签名 s 的第 i 部分位则由下式给出:

$$s_i = \begin{cases} x_{i,0} & \text{其中, } m_i = 0 \\ x_{i,1} & \text{其中, } m_i = 1 \end{cases}$$

很明显, 不能再次使用你的私钥, 因此, 这就是一次性签名的由来。验证者拥有公钥, 并进行检查:

$$\begin{cases} y_{i,0} = h(s_i) & \text{其中, } m_i = 0 \\ y_{i,1} = h(s_i) & \text{其中, } m_i = 1 \end{cases}$$

你将发现, 验证者需要附加的证据来进一步证实值 $y_{i,0}$ 和 $y_{i,1}$ 确实是公钥。我们将在 12.5.1 节回到这个主题。

此外, 除了依赖数学问题的难解性或者依赖其他的一些密码原语的强度, 还可以依赖抗损害硬件设备的难度。这个设备包括一个秘密的签字密钥和/或秘密的验证密钥。这个设备的构造可以保证它不能使用签名密钥来验证或者不能使用验证密钥来签名。为了签署一份文档, 设备可用它的签名密钥来构造 MAC, 然后把它附加到文档上。为了验证这个签名, 验证者的设备必须拥有签字者的签名密钥作为验证密钥, 并使用这个密钥来生成 MAC, 把它同接收到的签名进行比较。

11.4.2 ElGamal 签名和 DSA

ElGamal 签名方案(1985)用实例说明了签名不是用私钥加密的。设 p 是一个大的、适当选择的素数。设 g 是一个以 p 为模的 $p-1$ 阶整数。设 a 是用户 A 的私有签名密钥, 且 $y_a = g^a \bmod p$ 是对应的公共验证密钥。

假设要签名的文档是一个整数 m , $0 \leq m < p$ 。如果不是的话, 你也可以应用一个合适的哈希函数, 然后对文档的摘要进行签名。为了对 m 进行签名, 用户 A 选择了一个随机数 k , $0 \leq k < p$, 以使得 $\gcd(k, p-1) = 1$, 计算 $r = g^k \bmod p$, 并且解方程:

$$a \cdot r + k \cdot s \equiv m \bmod p-1$$

求解未知的 s 。整数对 (r, s) 对构成 A 对 m 的签名。验证者需要 A 的验证密钥 y , 并检查:

$$y^r \cdot r^s \text{ 与 } g^m \bmod p \text{ 是否相等}$$

对于一个正确的签名, 等式:

$$y^r \cdot r^s = g^{ar+ks} = g^m \bmod p$$

成立。这种方案的安全与离散对数问题紧密相关, 但是并不等价于离散对数问题。许多更安全和

更有效的签名方案都是从 ElGamal 签名方案派生出来的。一个这样的签名算法是数字签名算法 (DSA; 美国商务部, 美国标准与技术协会, 2000)。在这个方案中, 用户 A 的私钥和公钥通过下述步骤生成。

- (1) 选择一个素数 q , 满足 $2^{159} < q < 2^{160}$ 。
- (2) 选择一个整数 t , ($0 \leq t \leq 8$) 和一个素数 $p(2^{511+64t} < p < 2^{512+64t})$, 使得 q 能整除 $p-1$ 。
- (3) 选择 α , ($1 < \alpha < p-1$), 并计算 $g = \alpha^{(p-1)/q} \bmod p$ 。如果 $g=1$, 带入一个新的 α 再进行试算。(这一步计算以 p 为模的 q 阶的生成元 g 。)

(4) 选择一个 a , 使其满足 $1 \leq a \leq q-1$ 。

(5) 计算 $y = g^a \bmod p$ 。

(6) A 的私钥为数值 a , 公钥为 (p, q, g, y) 。

A 为了对文档 m 进行签名, 使用 SHA-1 计算哈希值 $h(m)$, 并且转换成一个整数, 然后:

- (1) 随机地选择一个整数 k , $1 \leq k \leq q-1$;
- (2) 计算 $r = (g^k \bmod p) \bmod q$;
- (3) 计算 $k^{-1} \bmod q$;
- (4) 计算 $s = k^{-1}(h(m) + ar) \bmod q$ 。

A 对 m 的签名就是整数对 (r, s) 。这个签名将用 A 的公钥 (p, q, g, y) 通过以下步骤进行检测:

- (1) 验证 $1 \leq r \leq q$ 和 $1 \leq s \leq q$;
- (2) 计算 $w = s^{-1} \bmod q$;
- (3) 计算 $u_1 = w \cdot h(m) \bmod q$ 和 $u_2 = r \cdot w \bmod q$;
- (4) 计算 $v = (g^{u_1} y^{u_2} \bmod p) \bmod q$ 。

当且仅当 $v = r$ 时成立。椭圆曲线数字签名算法 (elliptic curve digital signature algorithm, ECDSA) 与 DSA 相类似, 但是它是同椭圆曲线一起参与计算而不是以素数 p 为模的数。在 ANSI X9.62 标准中对 ECDSA 进行了说明。

11.4.3 RSA 签名

RSA 是用它的发明者 Rivest, Shamir and Adleman (1978) 的第一个字母命名的, 它同样可以用于签名和加密。RSA 的这种特定的性质必须对许多关于数字签名和公钥密码的流行的误解负责。在 RSA 签名方案中, 用户 A 选择两个素数 p 和 q , 以及一个私有签名密钥 e , 满足 $\gcd(e, p-1) = 1$ 和 $\gcd(e, q-1)$ 。公共验证密钥则由积 $n = p \cdot q$ 和指数 d 组成, 并且满足

$$e \cdot d \equiv 1 \bmod \text{lcm}(p-1, q-1)^\ominus$$

签名的文档是整数 m 。如果原始文档太长, 可以应用哈希函数和填充获得摘要进行签名。采用一个适当的哈希函数 h , 计算 $h(m)$, 以便使 $1 \leq h(m) < n$ 。从安全方面上讲, 这个哈希值是冗余信息, 验证者可用它来识别真正的文档。为了签名 m , 用户 A 用下式生成签名:

$$s = h(m)^e \bmod n$$

验证者需要 A 的验证密钥 (n, d) 并检查

$$s^d \text{ 与 } h(m) \bmod n \text{ 是否相等}$$

对于一个正确的签名来说, 这个等式是成立的, 因为

$$s^d = h(m)^{e \cdot d} = h(m) \bmod n$$

\ominus $\text{lcm}(p-1, q-1)$ 是欧拉商数。——译者注

RSA 安全与因式分解的难度式相关,但是并不等价于它的难度。任何一个执行 RSA 的安全都依赖于更多的因子。比如,上面给出的高级别的描述并没有解释 $h(m)$ 是怎样编码作成为一个模为 n 的整数。用 0 来追加一个 160 比特的 SHA-1 函数去获得一个 1024 位的整数将是一个错误的选择。编码函数选择得不好会引起脆弱性。当前推荐的执行 RSA 的方法叫做随机签名方案 (probabilistic signature scheme, RSA-PSS)。

11.5 加密

我们保留了术语加密 (encryption), 它表示保护数据机密性的算法。加密算法, 也叫做加密器 (cipher), 在加密密钥的控制下, 将明文 (plaintext 或 cleartext) 加密。我们用 $eK(X)$ 来表示由密钥 K 加密的明文 X 。用适当的解密密钥解密 (decryption), 可从密文中恢复明文。我们用 $dK(X)$ 来表示用密钥 K 解密的密文 X 。对于固定密钥, 一个确定的加密算法能得出相同的密文。在相同的密钥下, 一个随机加密算法可对相同明文的不同加密给出不同的密文。

有些加密算法提供了检测违反完整性的方法, 但情况并非总是这样。你甚至可能注意到签名算法被描述为使用私钥的加密, 但是, 这种说法常常是不正确的, 并且常带有误导性。

加密算法以两种形式出现。在对称加密算法中, 加密和解密使用相同的密钥, 这个密钥必须保密。所有参与者共享同一密钥, 他们可以阅读彼此的加密数据。为了对不同参与者建立专用信道, 每个信道均需要一个新的密钥。维护大量的共享密钥可能成为一种十分繁重的管理任务。

非对称加密算法 (asymmetric encryption algorithms), 也叫做公钥算法 (public key algorithm), 加密和解密使用不同的密钥。加密密钥可以公开; 解密密钥仍然是私有的, 必须保密。很明显, 这两个密钥在算法是相关的, 但是想要从公钥导出私钥必须是不可行的。为了区分对称密码系统和公钥密码系统, 我们在对称密码系统的上下文中使用密钥, 而仅仅在非对称密码系统的上下文中使用私钥。

在密钥加密系统中, 很明显, 安全管理任务就是将正确的密钥存放在合适的地方, 这种安全管理任务显而易见。而公钥密码系统看似可简化管理。毕竟, 公钥是公开的, 且不需要保密。当你使用公钥加密算法加密一份文档的时候, 可能需要知道谁能够阅读这个加密后的文档。更常见的是, 私钥可能授权给委托人或者用作携带访问权限的能力, 例如, 读一份文档。现在, 保证在公钥和访问权限或者与对应的私钥联系在一起的委托人之间的联系是一项主要的任务。在 12.5.1 节, 我们将回到这个主题。

加密算法也可以分成块加密器 (block cipher) 和流加密器 (stream cipher)。区分它们有两个标准。

- 块尺寸: 块密码加密较大的数据块, 典型的是使用一个复杂加密函数的 64 位块。块加密器的安全取决于加密函数的设计。流加密器加密较小的数据块, 典型的是位或者字节, 使用一个简单的加密函数, 例如, 位异或运算。正如你想象的一样, 这种区分在边界处就变得模糊不清了。一个 16 位的块仍然是一个大的块吗? 加密算法什么时候是简单的?

- 密钥流: 块加密器使用同一密钥来加密属于同一文档的所有的块。而流加密器借助连续变化的密钥流加密整个文档。流加密器的安全依赖于密码流生成器的设计。在这种定义下, 使用反馈模式的 DES (见下文) 属于流加密器范畴。

11.5.1 数据加密标准

数据加密标准 (Data Encryption Standard, DES) 是计算机安全和密码学发展史上的一个重要里程碑。DES 于 20 世纪 70 年代开发的, 作为美国政府保护非机密信息的标准, 并且作为联邦信

息处理标准发布(Federal Information Processing Standard; 美国商务部, 美国国家标准局, 1977)。DES 在 56bit 密钥控制下对 64bit 明文块加密。每一个密钥均通过一个奇偶检验字节扩展导 64bit 工作密钥。与大多数的块加密器一样, DES 基于 Feistel 原理。Feistel 加密器在大量的循环中迭代相同的基本步骤。第 i 轮的数据被划分成左半部分 L_i 和右半部分 R_i , 而输出采用下式计算:

$$L_{i+1} = R_i$$

$$R_{i+1} = L_i \oplus F(K_i, R_i)$$

其中 F 是某种非线性函数, K_i 是在这次循环中的子密钥, 如图 11-6 所示。这个操作的逆过程可以通过同样的线路来进行:

$$R_i = L_{i+1}$$

$$L_i = R_{i+1} \oplus F(K_i, L_{i+1})$$

在 DES 中的非线性函数 F 将 32bit 输入 R_i 扩展成 48bit 块, 并使用 48bit 子密钥 K_i 进行位异或运算。这个中间结果划分为 8 个 6bit 的块, 它们用作 8 个 S 盒(substitution box, 替换盒)的输入。每个 S 盒将 6bit 的输入转换成 4bit 的输出。 S 盒的输出通过 P 盒(permutation box, 置换盒), 它对 32bit 的输入执行位置换, 给出函数 F 的结果。

DES 要进行 16 轮(round)这样的运算。每轮都使用不同的 48bit 子密钥 K_i , 这个子密钥是从 56bit 的 DES 密钥衍生出来的。第一轮输入是由初始化置换 IP 处理的, 最后一轮的输出再进行一次逆置换, IP^{-1} 。图 11-7 给出了 DES 的示意图。我们省略了密钥编制表算法的细节, 扩展方案, S 盒和置换。

20 世纪 70 年代, 当 DES 变成了标准的时候, 人们预料它有 15 年的适用期。尽管 DES 正在被高级加密算法(AES; 美国商务部, 国家标准与技术协会, 2001)取代, DES 老当益壮, 仍然得到广泛使用, 特别在商务和金融领域。对 DES 安全的主要挑战不是来自新的密码技术, 而是它的密钥大小。今天, 不需要专用设备, 对 56bit 密钥空间的穷举搜索已经很容易了。工作站的性能正逐年增长, 网络使得业余密码分析者可以使用更多的资源。在这样对抗的环境下, 你只能期望一个 56bit 的密钥能够生存几个星期或者几个月, 而不是几十年或者几个世纪。多重密码可扩展密钥的大小, 而不必改变算法。推荐的选择是使用了三个 56bit 密钥的三重(triple)DES。因为它保持了与单一的 DES 的向后兼容性, 使用两个 56bit DES 密钥 K_1 和 K_2 , 按下式对明文 P 进行加密: $C = eK_1(dK_2(eK_1(P)))$ 。

2001 年, Rijndael 算法(Daemen and Rijmen, 1999)被用作 AES, 美国联邦的一个标准。这个算法可以使用大小为 128bit, 192bit 和 256bit 的密码。AES 使用 128 位的数据块工作。Rijndael 专门用来运行于 192 位和 256 位数据块之上, 其中数据块的长度可以被独立于密钥长度选择。

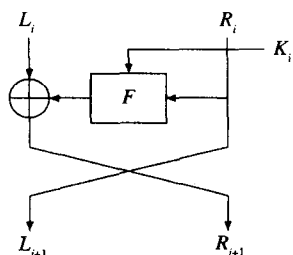


图 11-6 Feistel 原理

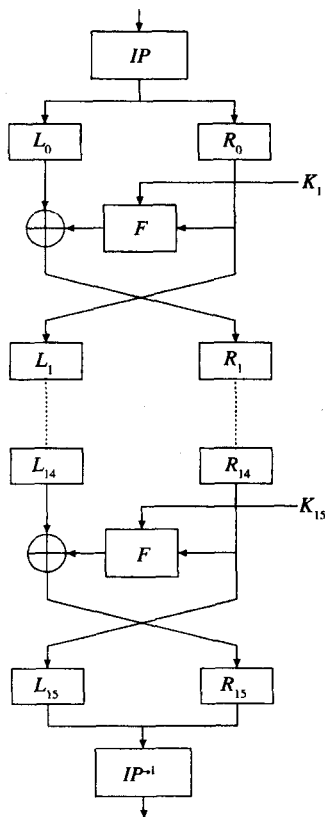


图 11-7 DES 算法

11.5.2 块加密器模式

块加密器可以用于各种各样的加密模式。在电子密码本 (electronic code book, ECB) 模式中, 每个明文块都用相同的密钥独立加密。这种模式可能泄漏关于明文的信息。如果一个明文块是重复出现的, 这种重复性也会出现在密文中。而且, 完整性保护非常有限。解密不检测密文块的顺序是否已经改变, 有些块是否已经丢失, 或者有些块是否已经被复制。

在图 11-8 的密码块链 (cipherblock chaining, CBC) 模式中, 前面的密码块 C_{i-1} 在加密前被加到 (位异或) 下一个明文块 P_i , 即:

$$C_i = eK(P_i \oplus C_{i-1})$$

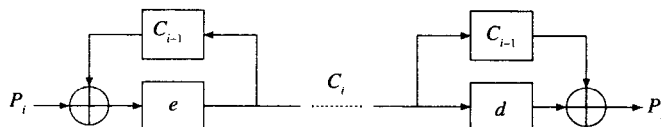


图 11-8 密码块链模式

因此, 重复的明文块不会作为重复的密文块表现出来。对于第一个明文块 P_1 , C_0 用作初始化向量。这个初始化向量通常是保密的, 尽管在很多的应用中, 这并不是必需的安全条件。对每一个消息, 初始化向量都应该改变, 以确保观察者不能检测到两个明文始于相同的块。初始化向量也被用来解密第一个密文块。密文块 C_i 使用下式进行解密:

$$P_i = C_{i-1} \oplus dK(C_i)$$

如果一个密文块被破坏了, 这种损坏将被限制在两个明文块中。假设 \tilde{C}_i 用来代替了 C_i , 在下列运算后:

$$\tilde{p}_i = C_{i-1} \oplus dK(\tilde{C}_i)$$

$$P_{i+1} = \tilde{C}_i \oplus dK(C_{i+1})$$

正常的解密服务又恢复了。

输出反馈模式 (output feedback, OFB; 参见图 11-9) 使用块加密器作为流加密器的密钥流生成器。在这个模式中, 明文可以以小于密码算法所要求的块尺寸的块为单位进行处理。加密函数的输入存储在寄存器中。该寄存器的初始内容由初始化向量来确定。为了加密一个明文块, 可把它添加到加密函数输出的一个子块中 (按位异或)。加密函数的输出则被反馈到了移位寄存器。解密过程与加密完全相同。每条消息对应的初始化向量均有变化, 但是不需要保密。在密文块 C_i 传送中的错误仅仅会影响对应的明文块 P_i 。因此攻击者可以通过在对应位置修改密文块来选择性地修改明文块。

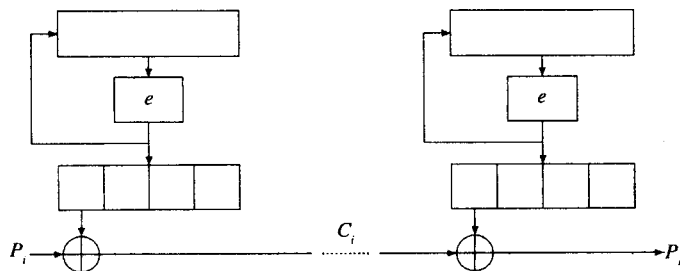


图 11-9 输出反馈模式

密码反馈(cipher feedback, CFB)模式(图 11-10)使用块加密器来生成与数据相关的密钥流。并且,明文也可以在比密码算法要求的尺寸小的情况下处理。在这种模式中,先前的密文块被反馈到一个移位寄存器。这个寄存器的内容被加密,且这个密文的子块被添加到下一个明文块中(按位异或)。解密过程与加密相同。在这种模式中,每条消息对应的初始化向量均有变化。初始化向量被要求用来解密第一个密文块,不必保密初始化向量。密码块的传输错误或者更改将影响解密,直到改变的块从移位寄存器中移出,并将加密函数反馈给接收端。

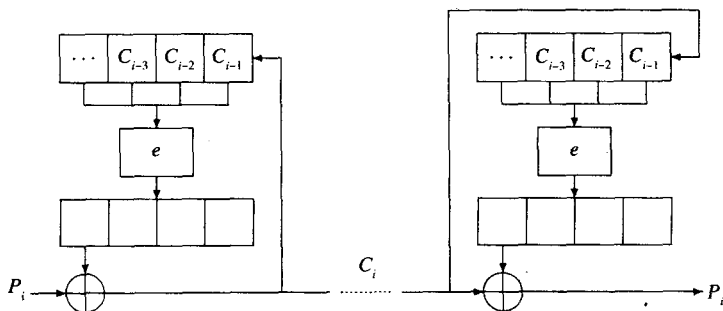


图 11-10 密码反馈模式

11.5.3 RSA 加密

从 RSA 签名方法中我们已经熟悉这个方案的基本结构了。当 RSA 用于公钥加密算法时,用户 A 选择两个素数 p 和 q , 一个私有解密指数 d , 满足 $\gcd(d, p-1) = 1$ 和 $\gcd(d, q-1) = 1$ 。公共加密密钥由积 $n = p \cdot q$ 和指数 e 组成, 满足:

$$e \cdot d \equiv 1 \pmod{\text{lcm}(p-1, q-1)}$$

消息必须被划分成块, 使得每个块都是一个小于 n 的整数。为了将消息 m 发送给 A , 发送者计算:

$$c = m^e \pmod{n}.$$

接收者 A 使用私有解密密钥 d 以获得:

$$c^d = m^{e \cdot d} = m \pmod{n}.$$

不要被这个算法的简单性所欺骗。正确的执行是相当机警的, 过于简单的“教科书似的 RSA”的执行有可能是**不安全的。

1. 填充

RSA 是一个块密码器。根据密钥的长度, 消息被分成 1024(或者 2048, 4096, ...) 比特长的块。当对一条消息进行加密的时候, 必须采用填充(padding)来使消息的长度达到多个块长度的倍数。填充可以防御一些攻击。当解密一条消息的时候, 接收者可以检查填充的数据并且丢弃语义上不正确的填充的明文。另一方面, 填充可能被攻击者利用。PKCS#1 v1.5 曾建议再填充一个数据 D , 如下所示:

00	02	PS	00	D
----	----	----	----	---

填充的前两个字节分别是 0 和 2。PS 是一个非 0 字节长度 $|n| - |D| - 3$ 产生的假随机数。这样一来, 数据 D 前面会有另外一个值为 0 的字节。

Bleichenbacher(1998)发现了一个针对带有这种填充方式的 RSA 的攻击: 如果接收者发信号表示解密是否失败或者成功, 这个攻击需要大约 2^{20} 可选择的密文来得到明文。这个攻击最典型

的设置将是 SSL 协议(参见 13.4 节)。重新得到的这个数据是一个会话密钥,接收者是一个服务器。攻击者截取了一个加密的会话密钥,然后向服务器发送了一个可选择的密文。当解密失败时,服务器回应一个错误信息。没发送错误信息表示已成功,并缩小了包含会议密钥的间距。攻击者一直重复这项工作, 2^{20} 次尝试后就可以确定唯一的密钥了(在密钥中,10 亿已经是个很小的数了)。

由于存在这种攻击,最佳非对称加密填充(Optimal Asymmetric Encryption Padding, OAEP)被采用来作为填充 RSA 消息的新标准。OAEP 很吸引人们的注意,因为它对自己的安全有正式的定理(Bellare and Rogaway, 1995)。然而,这个定理也有破绽,但是可以修复(Shoup, 2001)。与此同时,市上又会出现新的攻击(Manger, 2001),在写这本书时,关于填充安全计划的研究还在继续进行中。

2. 经验教训

即使加密算法在抽象的数学层面上是安全的,也可以找到对抗具体的执行的攻击。这样,就需要在抽象的低级别中的安全分析。

11.5.4 ElGamal 加密

在 ElGamal 公开密钥算法中, p 仍是一个大的且适当选择的素数,且设 g 是一个以 p 为模的高阶整数。设 a 是用户 A 的私有解密密钥,且 $y_a = g^a \bmod p$ 是相应的公共加密密钥。消息被划分成块,使得每个块都是一个小于 p 的整数。为了将一个消息块 m 发送给 A ,发送者选择一个随机数 k ,计算 $r = g^k \bmod p$,发送如下密文:

$$(c_1, c_2) = (r, my_a^k)$$

给 A 。 A 使用其私有解密密钥 a ,可以获得:

$$\frac{c_2}{c_1^a} = \frac{my_a^k}{r^a} = \frac{mg^{ak}}{g^{ak}} = m$$

在这种方案中,密文块长度是明文块的两倍。另一方面,如果两个明文块相等,相应的密文块却是不相同的。随机数 k 仅仅用于加密一次。随机数的重复使用会严重地削弱密码的强度。

11.6 密码机制的强度

度量密码算法的强度是一门不精确的艺术,有时建立在稳定的数学基础上,有时又依赖于直觉和经验,一个密码算法可以是:

- 经验安全(empirically secure)。
- 可证明的安全(provably secure)。
- 无条件安全(unconditionally secure)。

如果一个算法已经受住了时间的检验,那么我们说它是经验安全的。长期的分析没有发现它有什么严重的缺陷,虽然没有证明该算法不会最终受到一个新的或者有创造才能的攻击,算法已经在密码学界范围内被广泛接受。除去它的密钥长度,DES 便是一个经验安全的最好的例子。新的分析方法,比如微分密码分析学,已经加强而不是削弱了已知的 DES 安全。

可证明的安全算法,似乎提供了计算机安全一直渴望的东西,即可证明的安全。可证明的安全是在复杂理论的框架结构内表示的。如果破解一个算法至少和解决另外一个问题一样的困难,而另外一个问题我们已经知道它几乎是不可破解的,那么这个算法就是可证明安全的。这个定义听起来很好,但是阅读这些限制性的附属细则却很费劲。

“至少一样困难”是一个渐近的概念。对于“足够大”的问题实例它是成功的。然而，这个理论不会告诉你什么是足够大的。相反，你必须评估目前的计算设备的能力和算法设计的进展情况。例如，我们能够进行因式分解的数的大小在近几年中不断地增长。你会同意这是一个经验论点。甚至还有更糟的情况。密码系统特别偏爱的难解问题是因式分解和离散对数问题。现在并没有实际证明这些问题一定是难解的。密码系统再次依赖于经验论证：到目前为止没有找到快速算法，因此极不可能有大的突破。但从乐观的角度来说，这一理论也产生了很多建设性的结果，为攻破与破解其他密码原语难度相关的密码方案所需的工作量确定了下界。

可证明的安全算法有可能被拥有足够多计算资源的攻击者攻破。当然，你可以祈祷这些必需的资源超出了任何攻击者的能力。无条件安全算法不能被攻破，即使攻击者拥有无限的计算能力。无条件安全是按照信息论的概念来表示的。如果一个攻击者查看密文没有获得关于明文的附加信息，则这个算法就是安全的。

无条件安全算法的标准实例是一次性填充(one-time pad)。发送者和接收者共享一个真正的随机密钥流，这些密钥只使用一次。密文是明文和密钥流的位异或运算的结果。接收者采用相同的密钥流对密文进行位异或运算，这样便可以从密文中恢复出明文：

$$\text{密文} \oplus \text{密钥流} = \text{明文} \oplus \text{密钥流} \oplus \text{密钥流} = \text{明文}$$

因为每个密钥都是等概率的，攻击者不可能猜测到关于明文的任何信息，而且在看到密文以前也不可能对明文有任何的猜测。

注意！即使无条件的安全密码已经被破解。当操作者抄近路和两次使用相同的密钥流时，攻击者若叠加这两个密文将看到两个明文的组合：

$$\text{密文}_1 \oplus \text{密文}_2 = \text{明文}_1 \oplus \text{密钥流} \oplus \text{明文}_2 \oplus \text{密钥流} = \text{明文}_1 \oplus \text{明文}_2$$

理解覆盖在一起的两个明文消息完全不是一个困难的密码分析问题。Venona 工程证明这样的事件甚至发生在冷战的最严重时期。

最后再强调一个至关重要的事实。时常，密码系统被攻破是因为劣质的密钥管理而不是算法的固有弱点。在第二次世界大战时的 Enigma 就是说明这种观点的最著名的例子。因此，密钥管理协议的安全就显得极端的重要。第 12 章中就包含了密钥管理协议的说明。

11.7 演示

总结 Preneel et al. (2003) 报道的一些演示数据，不难发现其中有许多算法和平台的细节研究，来显示加密算法的执行性能。我们采用的方法是在一台奔腾 III 的电脑上，在 Linux 的环境下，用软件运行这个算法。不同的编译器其执行的次数也不同。表 11-1 给出了哈希函数、流加密器和块加密器的执行度量，作为每个字节的指令周期数。表 11-2 比较了 RSA 和两个 ECDSA 变量。度量以每次调用的周期数(百万级)给出。RSA 加密和 RSA 签名用公共指数 $e = 3$ 来演示使用短型公共指数获取的最大限度的收益。给出算法的密钥长度作为参考。

表 11-1 哈希函数和对称密钥算法的性能测定

运算法则	周期/字节	运算法则	周期/字节
RC4	7-8	SHA2-512	83
MD5	7-8	Rijindael-128	25-30
SHA-1	15	DES	60

表 11-2 非对称算法和数字签名算法的性能测定

运算法则	运算	周期/调用	密钥设置(周期)	密钥长度(比特)
RSA_OAEP	加密	2.026M	1.654M	1024
	解密	42.000M		
RSA_PSS	签名	42.000M	1.334M	1024
	验证	2.029M		
ECDSA_GF(p)	签名	4.775M	4.669M	160
	验证	6.085M		
ECDSA_GF(2^n)	签名	5.061M	4.825M	163
	验证	6.809M		

11.8 深层阅读

如果你对秘密通信的历史感兴趣, *Kahn(1976)* 的著作算得上是一本比较好的参考书。最新的关于密码系统的专业的参考可以阅读 *Menezes, van Oorschot and Vanstone(1997)*。 *Schneier(1996)* 的著作给出来对现代密码学的比较好的、非数学的介绍, 它还有一个非常丰富的参考目录, 并收集了大量的密码算法。在这些书中, 你可以找到在本章中提到的算法的所有细节和深层的信息。 *Differ and Hellman(1976)* 是公开密钥算法中, 素数是一个重要的因数。可以在 *Granville(2005)* 中找到有关原始测试的优秀阐述。有关新的块加密器模式的发展, 可参阅 NIST 专门的出版物 800-38A, 800-38B 和 800-38C。有关 Venona 文件可以访问 <http://www.nsa.gov/venona/>。

11.9 练习

练习 11.1 密码协议被用于部门在不安全的网络上进行安全通信。这种说法正确吗?

练习 11.2 密码学需要物理安全。这句话在什么程度上是正确的?

练习 11.3 假定要发起一个需要估算 2^{80} 次运算的哈希函数, 哈希值应该多长才能分别达到弱和强的冲突对抗。

练习 11.4 就 DSA 写出使条件 $v=r$ 成立的有效签名。

练习 11.5 给定需要 n^3 次操作的 n 位比特整数的一个模指数运算, 把 1024 比特移到 2048 比特的 RSA 会造成多大程度的执行影响?

练习 11.6 考虑 RSA 是一个没有哈希函数的签名算法, 例如 $s = m^e \bmod n$ 。请解释如果消息 m 上没有正式的检测, 攻击者如何才能伪造签名, 以及到哪种程度。

练习 11.7 当一个文档因太长而不能被 DSA 直接处理时, 对文档计算哈希值, 然后对哈希值进行签名。你要求这个哈希函数具有什么样的性质, 才能阻止攻击者伪造签名?

- 区分两种情况, 一种是攻击者仅知道受害者签名的消息, 另一种是攻击者可以选择受害者会签名的消息。
- 区分选择性伪造和存在性伪造, 在选择性伪造中, 攻击者对伪造消息有控制权, 而在存在性伪造中, 攻击者对伪造消息没有控制权。
- 考虑特定的使用可逆转签名算法(如 RSA、哈希函数)的需求。

练习 11.8 是否能够用任意的“素数对”安全地使用 RSA? 调查使用强素数的原因, 以及强素数的属性。当素数更长时, 这些原因是否继续有效?

练习 11.9 在 ElGamal 签名算法中, 如果随机值 k 用于两个不同的文档中, 试说明私有签名密钥可以如何被泄密?

练习 11.10 NP-complete 问题是否是构建密码算法的合适基础?

第 12 章 分布式系统中的认证

密码学可把安全问题转换为密钥管理问题。为了使用加密、数字签名或是消息认证码 (MAC), 参与者必须持有正确的密钥。在公钥算法中, 参与者需要拥有可信公钥。在对称密钥算法中, 参与者需要共享密钥。这可以通过邮件送信来实现——一个常见的信用卡分发 PINs 的方法, 或者通过信使在地点之间往返和递送密钥来实现。这些建议都是非常不可靠或者是相当昂贵的。理想的情况是, 我们能够在现有的通信基础框架上执行密钥管理机制。

当双方商谈一个新密钥的时候, 因为这是它们第一次通信或是它们开始了一个新的会话, 此时它们可能都必须证明它们是谁。证明身份和建立密钥曾经都叫做认证。本章将讨论并区分这两个行为。

目标

- 澄清认证可能的含义。
- 阐述一些主要的密钥建立协议。
- 讨论怎样将公钥链接到用户身份认证上。
- 说明密码协议如何应用到计算机安全中。

12.1 引言

公钥推算起来比对称密钥花费更昂贵。花费的原因包括推算时间和带宽。这两者都取决于密钥的长度。用长期密钥来减少“攻击面”更为可取。这可以预防那些需要收集大量加密材料的攻击。作为这两个问题的一个解决方法, 可以使用长期密钥来建立短期的会话密钥 (session key)。

将密钥的使用限制于一个专门的目的不失为很好的密码学实践。在密钥管理中, 我们可能会用到密钥加密密码和数据加密密码。密钥使用 (key usage) 的例子有加密、解密、数字签名 (用于认证) 和通信安全中的认可等。在公钥加密中, 我们强烈推荐不要对加密和数字签名使用一个单一的密钥对。密钥使用的其他例子为: 在分等级的密钥管理方案中掌握密钥和交易密钥。

12.2 密钥建立和认证

建立会话密钥协议曾被称为认证协议。毕竟, 它们的目的也是让你知道“你在和谁谈话”。在文献中, 特别是在旧的著作中, 你可能仍然可以找到这些协定。今天, 密码学中的术语区分了认证和密钥建立。从认证逐步分离出密钥建立在国际标准化组织的发展过程中有所体现。20 世纪 80 年代, ISO/OSI 框架 (ISO 7498-2; 国际标准化组织, 1989) 仍然有面向会话的实体认证观点。

对等实体认证 确认连接中的对等实体就是其所声名的身份。在创建连接的数据传输阶段时或者期间, 这项服务被用来确认一个或多个相连实体的身份。

在 19 世纪 90 年代早期, 国际标准化组织 ISO/IEC 9798-1 (国际标准化组织, 1991) 用一种方法定义了实体认证, 该方法不再包含安全会话的建立。

实体认证 实体认证机制允许一个实体核实另一个实体声称的身份。实体认证仅能在

认证交换的例子中被确定。

在该解释中, 实体认证就是核实一个实体是否是活动的。在通信网络中, 该属性与对方失效探测 (dead peer detection) 相关联。

在单方认证中, 仅一方实体被认证。在相互认证中, 实体双方都要被认证。

12.2.1 远程认证

任何时候想改变的环境, 都必须重新评估已建立的安全机制是否适合。从一个集中式系统中移到一个分布式系统显然对安全会有很大影响。想要看看这个改变对你的影响有多大, 可以参看通过口令的认证 (第3章)。当用户在与一台主机有固定连接的终端上工作时, 口令是很有用的。这里, 你可以有充分的理由相信在终端和主机之间的连接是安全的, 并且窃听口令, 更改或者插入信息, 或者控制一个会话都是不可能的。而在一个分布式设置中, 这种关于通信连接安全方面的基本假设不容易被证实。

在网络上无保护的口令传送显然是一个漏洞, 很容易利用该漏洞。口令嗅探器 (password sniffer) 是程序, 这些程序监听网络通信量, 并且选出包含口令和其他安全相关信息的数据包。口令仍是一个在分布式系统中受欢迎的认证机制。这里以 http 协议作为例子。它在客户机和服务器上运行。客户端发送 http 请求给服务器。为了验证这个客户端, http 协议有一个基于客户和服务器都知道的共享密钥 (口令) 内置认证机制。有两种认证, 基本访问认证 (Basic Access Authentication) 和摘要访问认证 (Digest Access Authentication)。

在基本访问认证中, 客户端必须提供口令。当客户端请求一个受保护的资源, 服务器就回复 401 未授权的回应码。然后客户端将认证信息 (基本的 64 位编码密码) 发送给服务器, 服务器再核实客户端是否已被授权访问资源。所有的认证数据都以明文方式发送。协议像这样运行。

- 客户端: GET/index.html HTTP/1.0
- 服务器端: HTTP/1.1 401 Unauthorized
WWW-authenticate Basic realm="SecureArea"
- 客户端: GET/index.html HTTP/1.0
Authorization: Basic am9ldXNlcjphLmIuYy5E
- 服务器端: HTTP/1.1 200 Ok (以及被请求文档)

相反, 分类访问认证并不以明文发送口令。在一个质询 - 回应 (challenge-response) 协议中使用了一个加密 hash 函数 h (一般是 MD5)。由服务器发送的 WWW 认证参数包括一个独立的质询, 即所谓的序列号 (nonce)。服务器负责保证每一个 401 回复都伴随一个独立的、先前没使用过的序列号值。客户端用一个授权的回应回复, 这个回应包括了明文用户名、刚接收到的序列号值以及所谓的“请求 - 摘要” (request-digest), 计算如下:

请求 - 摘要 = $h(h(\text{用户名} \parallel \text{领域} \parallel \text{密码}) \parallel \text{序列号} \parallel h(\text{方法} \parallel \text{digest-uri}))$

其中“digest-uri”涉及所请求的 URI 和给予 http 请求方法的“方法”。在 RFC 2617 中详细给出了这个协议的所有选项的详细描述。

术语序列号是由 Needham and Schroeder (1978) 提出的, 用来表示只被使用一次的独立值。序列号在协议设计中扮演很重要的角色。序列号可以是一个计数值、一个时间戳或是一个随机数。序列号没有必要一定是无法预知的, 但一些协议要求使用无法预知的序列号。它根据不同的安全目标来决定应该需要哪种类型的序列号。

RADIUS 协议 (远程认证拨入用户服务器, RFC 2865) 就是一个通过口令来实现远程用户认证的例子。与 http 摘要访问认证相似, RADIUS 包含了一个挑战 - 回应选择, 其中口令不会以明文方式传送。

12.2.2 密钥建立

为两个或更多的参与者建立以后共同使用的密钥的过程现在叫做密钥建立 (key establishment)。这个过程包括愿意建立共享密钥的双方, 通常叫做主角 (principal), 另外可能还有像认证服务器那样的第三方。在密钥建立协议中的主角可以不用和访问控制 (第 4 章) 中的主角一样。当第三方能够操作协议的安全目标时, 它叫做可信第三方 (trusted third party, TTP)。当主角调用它的服务时, 它们必须信任 TTP。

有时, 对一个密码学协议的数学细节分析揭示了存在这样一个弱密钥 (weak key) 的子集, 这个子集将允许入侵者进行欺骗。在设计一个密钥交换协议的时候, 你应该回答下面两个问题。

- 如果建立了一个弱密钥, 哪一方参与者会受攻击?
- 哪些参与者可以控制密钥的选择?

如果内部人员的误用行为能够影响密钥的产生, 使得能够选择一个弱密钥, 这便给内部人员攻击留出了余地。这种情况被认为是密钥控制。因此, 密钥建立服务可以根据每一个主角对新密钥和提供的实际安全保证所作的贡献进一步区分开来 (Menezes, Oorschot and Vanstone, 1997)。

- 密钥传输: 参与者一方创建一个秘密值, 然后把它安全地传送给另一方 (另外几方)。
- 密钥一致: 双方都参与秘密值的生成以便没有哪一方能够预知输出的结果。
- 密钥认证: 参与者一方确信除了专门确认过的第二参与方之外, 没有其余的参与方能够访问特定的密钥。
- 密钥确认: 参与者一方确信第二参与方 (可能无法辨别其身份) 拥有一个特定的密钥。
- 显式密钥认证: 双方都持有密钥认证和密钥确认。

图 12-1 显示了实体认证的意义是如何随着时间的推移发展的。评估一个协议时所考虑的其他因素是第三方要求。它包括 TTP 吗? 它是离线的还是在线的?

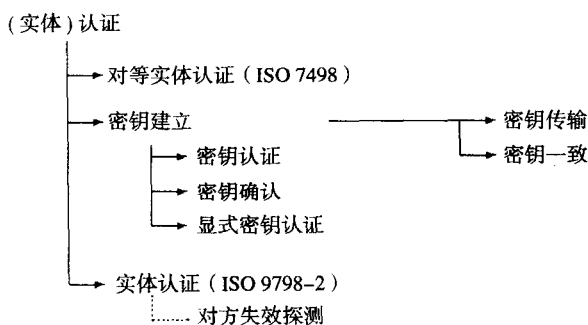


图 12-1 认证和密钥建立的术语

12.3 密钥建立协议

为其他协议使用而建立密钥的密码协议称为密钥建立协议。今天, 研究文献提供了对这些协议的更广泛的选择。我们选择下面的这些协议来阐述重要的设计技术。

12.3.1 认证密钥交换协议

在我们的第一个协议中, AKEP2 (认证密钥交换协议 2; Bellare and Rogaway, 1994), 用“廉

价的”哈希函数代替加密，并且不依赖于 TTP。两个主角 A 和 B 共享两个长期的对称密钥 K 和 K' ，并在各个协议中运行产生最新的随机数(序列号)，分别是 n_a 和 n_b 。该协议使用一个密钥哈希函数(MAC) h_K 和一个密钥单向函数 $h'_{K'}$ 。AKEP2 是一个三步协议。

- (1) $A \rightarrow B: n_a$
- (2) $B \rightarrow A: B, A, n_a, n_b, h_K(B, A, n_a, n_b)$
- (3) $A \rightarrow B: A, n_b, h_K(A, n_b)$

在第一步， A 发送一个质询 n_a 。在第二步中， B 用 $h_K(B, A, n_a, n_b)$ 回应并且发送自己的质询 n_b 。共享密钥是 $k = h'_{K'(n_a)}$ 。在第三步中， A 对此质询回应 $h_K(A, n_b)$ 。AKEP2 提供了相互的实体认证和(隐式)密钥认证。

12.3.2 Diffie - Hellman 协议

Diffie- Hellman 协议(1976)是一个密钥一致协议。主角 A 和 B 预先不共享一个密钥。他们选择一个大的可适当选择的素数 p 和一个以 p 为模的高阶元素 g 。主角 A 选择一个随机数 a 并且发送 $y_a = g^a \bmod p$ 给 B 。主角 B 选择一个随机数 b ，并且发送 $y_b = g^b \bmod p$ 给 A ，再计算 y_a^b 。收到 y_b ， A 使用它自己秘密的 a 来计算 y_b^a 。这是因为

$$y_a^b = g^{ab} = g^{ba} = y_b^a$$

参与双方都共享密钥 $g^{ab} \bmod p$ 。这个协议的安全性取决于离散对数问题(参见 11.2 节)的难度。一个能够计算离散对数的攻击者可以从 $g^a \bmod p$ 和 $g^b \bmod p$ 中获得 a 和 b 。Diffie- Hellman 协议的安全性是否同等于离散对数问题的安全性目前还不清楚。

1. man-in-middle 攻击

还剩下一个细小的问题。任何参与方都不知道自己是与谁共享秘密的。这个问题可能被处于 man-in-middle 攻击模式中的攻击者 M 利用。攻击者将自己插入 A 和 B 的通信路径之上，当 A 发起一个协议运行时回复 A ，并且在同一时间， M 假扮 A ，发起和 B 的协议运行(图 12-2)。主角 A 和 B 可能认为它们已建立了一个共享密钥，但实际上是 M 和 A 共享密钥 $g^{ax} \bmod p$ ，和 B 共享密钥 $g^{bx} \bmod p$ ，它扮演中继站的角色，能读取所有 A 和 B 之间的信息。

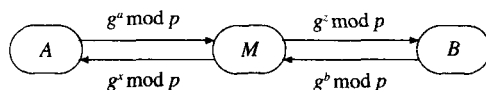


图 12-2 运行 Diffie- Hellman 协议下的 man-in-the-middle 攻击

2. station-to-station 协议

为了确保 Diffie- Hellman 协议的安全性，必须添加认证。station-to-station 协议(STS)(Diffie, van Oorschot and Wiener, 1992)按如下方式执行。除了 Diffie- Hellman 交换建立了一个共享会话密钥 $K: = g^{ab} \bmod p$ 以外，协议还使用加密算法和签名算法。未指定专门的算法。在下面的描述中， S_a 和 S_b 是 A 和 B 的签名密钥， sS_a 和 sS_b 表示在这些密钥下产生的签名。协议中的步骤如下：

- (1) $A \rightarrow B: g^a$
- (2) $B \rightarrow A: g^b, eK(sS_b(g^b, g^a))$
- (3) $A \rightarrow B: eK(sS_a(g^a, g^b))$

在第一步中， A 如同以前一样发起一个 Diffie- Hellman 密钥的交换过程。收到 g^a 后， B 选择一个随机数 b 并计算出会话密钥 $K: = g^{ab} \bmod p$ 。第二步之后， A 能够计算 Diffie- Hellman 密钥

交换的一部分并且得到 K ，然后解密 B 的消息的第二部分，核实 B 的签名。在最后一消息之后， B 能够解密和核实 A 的签名。STS 是一个密钥一致协议，此协议能够提供相互的实体认证和显式的密钥认证。

12.3.3 Needham-Schroeder 协议

Needham-Schroeder 协议 (1978) 是一个密钥传输协议。两个参与方 A 和 B 从一个服务器 S 中获得会话密钥。两个主角预先与服务器共享一个密钥。采用对称加密器进行加密。在消息中包含序列号 (随机质询) 以防止重放攻击。在协议描述中用到的约定如下：

- K_{as} : 由 A 和 S 共享的一个密钥。
- K_{bs} : 由 B 和 S 共享的一个密钥。
- K_{ab} : 在 A 和 B 之间使用的、由 S 创建的一个会话密钥。
- n_a, n_b : 分别由 A 和 B 产生的序列号。

图 12-3 显示了当实体 A 从服务器 S 中请求一个用于和 B 交流的会话密钥 K_{ab} 时产生的步骤。在这个协议的前三个步骤中， A 从 S 中获得了会话密钥，并将密钥传送给 B 。通过核实在服务器信息中返回的 nonce n_a ， A 能够确认这个会话密钥在回应它最近的请求时已经发送出去，而并不是前面一个协议运行中的重放。在最后两步中， B 验证了 A 当前在使用同样的会话密钥。在第 4 步和第 5 步中， A 执行了 B 的一个单方实体认证。

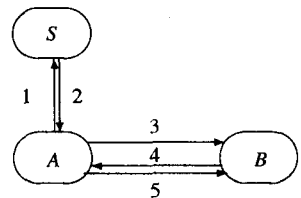


图 12-3 Needham-Schroeder 协议中的信息流程图

- (1) $A \rightarrow S: A, B, n_a$
- (2) $S \rightarrow A: eK_{as}(n_a, B, K_{ab}, eK_{bs}(K_{ab}, A))$
- (3) $A \rightarrow B: eK_{bs}(K_{ab}, A)$
- (4) $B \rightarrow A: eK_{ab}(n_b)$
- (5) $A \rightarrow B: eK_{ab}(n_b - 1)$

3. Denning-Sacco 攻击

如果只考虑单个协议运行，并且长期密钥 K_{as} 和 K_{bs} 未被泄漏过，那么在这种标准的假设下，Needham-Schroeder 协议实现了自己的目标。*Denning and Sacco* (1981) 发现了一个重放攻击 (replay attack)，即敌方 M 假扮 A 重新使用一个已泄漏的会话密钥 K_{ab} 。对手在协议的第 3 步开始行动并且在建立好会话密钥时，向 B 重放来自协议运行的第三方消息。 B 解密消息并且重新使用已泄漏的会话密钥：

- (3) $M \rightarrow B: eK_{bs}(K_{ab}, A)$
- (4) $B \rightarrow M: eK_{ab}(n'_b)$
- (5) $M \rightarrow B: eK_{ab}(n'_b - 1)$

这是一个已知的密钥攻击 (known key attack)，该攻击使用一个已泄漏的旧会话密钥来泄漏一个以后的会话。至于 B ，Needham-Schroeder 都不会提供密钥更新。

(从一个参与方的观点来看) 密钥如果能够保证是全新的，那么它就是新鲜的 (*Menezes, van Oorschot and Vanstone, 1997*)。

密钥更新有助于阻止重放攻击。

4. 完美的前向保密

当分析协议的时候，将过去的会话密钥的泄漏与长期密钥的泄漏区别对待，这样比较好。当

长期密钥泄密时, 就不能再保护以后的会话了。然而, 我们仍然希望过去的会话保持安全性。如果会话密钥或者长期密钥的泄密不会泄漏过去的会话密钥, 那么这个协议就达到了完美的前向保密(perfect forward secrecy)。术语“完美的前向保密”表示旧的会话密钥的秘密会一直延续使用。

12.3.4 基于口令的 password-based 协议

在 Needham-Schroeder 协议中, 当客户和服务器发起一个协议运行时, 它们就已共享了密钥。因此, 两个系统都必须配置成可安全存储密钥的形式。如果客户端是经由一个“不可信”的装置访问服务器的一个人, 那么我们只能依赖用户能记住的共享秘密, 即, 我们又回到了有关口令的问题上来。然后, 我们用一个口令 P 来加密一个随机产生的会话密钥 K_s , 然后用这个会话密钥来加密以后的数据:

(1) $A \rightarrow B: eP(K_s)$

(2) $B \rightarrow A: eK_s(\text{数据})$

这个本地协议有一个问题, 它很容易受到离线字典攻击。攻击者会猜测口令 P , 解密第一个信息并且得到一个候选的会话密钥 K_s' , 这个密钥用来解密第二个消息。当出现了一些有意义的文本时, 口令可能已经被正确破译了。

加密密钥交换(Encrypted Key Exchange, EKE)协议可以避免这个问题(Bellare and Merritt, 1992)。它使用对称加密算法用口令 P 作为密钥对数据进行加密, 并且也用到了一个公钥的加密系统。在协议运行中, 主角 A 产生了一个随机公钥/私钥对 K_a, K_a^{-1} 。在第一个消息中, A 发送在 P 下加密的(对称加密)公钥 K_a 给 B 。在第二个消息中, B 将随机产生的会话密钥 K_s 发送给 A , 首先在 K_a (公钥加密)下加密, 然后在 P 下加密(对称加密)。

(1) $A \rightarrow B: eP(K_a)$

(2) $B \rightarrow A: eP(eK_a(K_s))$

作为练习, 请说明这个协议不易受到离线字典攻击。针对每一个协议运行产生新的密钥对并不只是浅显的练习。下一步的工作是设计更有效的基于口令的密钥建立协议和对它们的形式化分析; 你可以参看 Halevi and Krawczyk(1999)的著作。

12.4 Kerberos

Kerberos 系统是美国麻省理工学院(MIT)在 20 世纪 80 年代的雅典娜(Athena)项目中开发出来的。雅典娜为 MIT 的校园学生提供了计算资源, 但是它的范围已经超出了 MIT 的校园范围, 包括了一些附加的管理功能, 比如财会功能。由 Kerberos 声明的危险和威胁在 Miller et al (1987)的著作中已有描述:

这种环境不适合敏感数据或者高风险的操作, 如银行事务处理、机密的政府数据、学生成绩、对危险实验的控制, 等等。这些风险主要有: 不能控制非授权的当事人使用资源、对系统或用户资源完整性的破坏, 以及大规模地侵犯个人隐私, 比如偶然地浏览所有的个人文件。

自此, Kerberos 被广泛地接受了。有几个工业标准已经采用 Kerberos 用于分布式系统的认证, 值得注意的是 RFC 1510。在分布式系统中, Kerberos 按服务认证客户(client)。认证是围绕票据(ticket)和集中式安全服务器的概念建立的。

Kerberos 来自于 Needham-Schroeder 密钥交换协议(参见 12.3.3 节)。像 DES 这样的对称密码系统常常被用来加密。用户名和口令用来认证用户, 但是口令不能通过网络进行传输。RFC

1510 给出了 Kerberos 版本 5 的一个详细规范说明, 包括当一个协议运行不能继续时发出的出错信息等。下面的简单描述省略了在 Kerberos 消息中的一些数据域, 并且仅仅处理了协议运行成功完成时的情形。

Kerberos 涉及了在客户机 C 上的用户 A 、服务器 B 和认证服务器 S 。用户 A 和认证服务器 S 共享一个密钥 K_{as} 。这个密钥是从用户的口令通过单向函数计算得到的。 K_{bs} 是一个被 B 和 S 共享的密钥, K_{ab} 是由 S 创建的在 A 和 B 之间使用的会话密钥, n_a 是由 A 产生的 nonce, T_a 是关于 A 时钟的时间戳。对于 B 使用的票据是 $\text{ticket}_B = eK_{bs}(K_{ab}, A, L)$, 其中 L 是票据的生命期(使用期限)。其核心工作如下。

- (1) $A \rightarrow S: A, B, n_a$
- (2) $S \rightarrow A: eK_{as}(K_{ab}, n_a, L, B), \text{ticket}_B$
- (3) $A \rightarrow B: \text{ticket}_B, eK_{ab}(A, T_a)$
- (4) $B \rightarrow A: eK_{ab}(T_a)$

为了开始一次会话, 用户 A 通过输入用户名和口令在客户机 C 上登录, 他输入用户名和口令。客户机将 A 的认证请求发送给 B 和 S 。第一条消息包含了 A 的标识、服务器的名字和 nonce, 所有都以明文形式发送。在第二步中, S 在自己的数据库中查询 A 的密钥 K_{as} (如果 S 不知道 A , 协议将停止), 产生会话密钥 K_{ab} 和票据 ticket_B 。 S 以在 K_{as} 下加密的数据结构形式将会话密钥发送给 A , 并将票据发送给 A 。在客户端, 密钥 K_{as} 从 A 的口令得到, 并且用于解密回复的第一部分内容。 A 得到会话密钥 K_{ab} 并核实 nonce。在第三步中, A 发送票据和认证符 $eK_{ab}(A, T_a)$ 给 B 。 B 用 K_b 解密票据, 并且获得会话密钥 K_{ab} 。 B 核对票据里面的标识符号是否和认证符匹配, 票据是否过期, 时间戳是否有效。时间戳的有效期必须考虑 A 和 B 的当地时钟的偏差。在第四步中, B 将在会话密钥 K_{ab} 下加密的时间戳 T_a 返回给 A 。

Kerberos 使用几个票据授予服务器 (ticket-granting server, TGS) 与一个认证服务器联合部署而成。Kerberos 认证服务器 (Kerberos Authentication Server, KAS) 认证主角登录并发放票据, 票据通常对一个登录会话是有效的, 并且能够从 TGS 获得其他票据。认证服务器有时也叫做密钥分发中心 (Key Distribution Center, KDC)。TGS 发放赋予主角访问网络服务的所需认证的票据。图 12-4 显示了在包含一个 KAS 和一个 TGS 的协议运行时所产生的步骤。这里, $K_{a,igs}$ 是 KAS 创建的用于 A 和 TGS 之间的会话密钥。这里有第二个 nonce n'_a 和时间戳 T'_a 。 L_1 和 L_2 是两张票据的生命期。票据授予票据 (ticket-granting ticket, TGT) $\text{ticket}_{A,TGS}$ 构造为:

$$\text{ticket}_{A,TGS} = eK_{igs}(K_{a,igs}, A, L_2)$$

其中 eK_{igs} 是由 KAS 和 TGS 共享的密钥。

- (1) $A \rightarrow KAS: A, TGS, n_a$
- (2) $KAS \rightarrow A: eK_{as}(K_{a,igs}, n_a, L_1, TGS), \text{ticket}_{A,TGS}$
- (3) $A \rightarrow TGS: \text{ticket}_{A,TGS}, eK_{a,igs}(A, T_a), B, n'_a$
- (4) $TGS \rightarrow A: eK_{a,igs}(K_{ab}, n'_a, L_2, B), \text{ticket}_B$
- (5) $A \rightarrow B: \text{ticket}_B, eK_{ab}(A, T'_a)$
- (6) $B \rightarrow A: eK_{ab}(T'_a)$

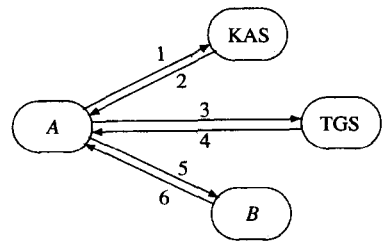


图 12-4 Kerberos 认证协议中的消息流

12.4.1 领域

KAS 是 Kerberos 领域的核心。一个 Kerberos 领域 (realm) 就是一个单一的管理范围, 控制着对一个服务器集合的访问。为了准备和运行 Kerberos, 主角必须向 KAS 注册, TGS 必须接收访问控制信息, 所有需要的密钥必须由安全管理员存放在适当的位置。

Kerberos 具有集中式安全系统所有的优点。单个安全策略只能由有限数量的安全服务器实施。因此, 检查系统安装时是否遵守了安全策略以及在需要的时候作某些变更, 还是相当容易的。

一个领域常常对应一个单一的组织。为了在其他组织中方便地访问服务器, 我们需要内部领域的认证。这需要认证服务器在不同领域中有“信任关系”。在这种情况下, “信任”是一个可共享的密钥。在组织间, 密钥共享建立在契约协议上。信任是可传递的吗? 如果在领域 R_1 和 R_2 之间有信任, 在 R_2 和 R_3 之间有信任, 那么在 R_1 中的客户端有权连接到领域 R_3 的服务器吗? 这里没有一个统一的答案。输出结果取决于领域之间的优先协议。

比如, 试考虑领域 R_1 中的用户 A , 它需要从自己的认证服务器 KAS_1 得到一个到领域 R_3 中的服务器 B 的票据。用户或者一些当事人服务机构发现, KAS_1 与 KAS_2 有一个可信任的关系, KAS_2 和 KAS_3 有一个可信任的关系, 这些信任关系是可传递的。应 A 的要求, KAS_1 为领域 R_2 产生一个 TGT, 并将这个 TGT 连同 A 的要求一起传送给 KAS_2 。依次地, KAS_2 为领域 R_3 产生一个 TGT, 并且将这个 TGT 连同 A 的要求一起发送给 KAS_3 。 KAS_3 为 B 创建票据, 并将票据发送给 A 。当请求从 B 获得一项服务的时候, A 所登录的客户端出示这张票据。

12.4.2 Kerberos 和 Windows

Kerberos 已经成为 Windows 中备选的认证协议。Windows 域对应于 Kerberos 领域。域控制者扮演 KDC 的角色。运行 Kerberos 的主体可能是用户也可能是机器。在 Windows 中, 认证是随后访问控制决策的基础。在 Windows 中, 访问控制的主角是 SID。(在此, 我们最终发现“主角”的两个定义之间有冲突。)因此, 一个主角的 SID 必须存储在票据中。Kerberos 票据, 按照在 RFC 1510 中的定义, 包含有用作客户端名字的强制性域的客户端名 $cname$ 和一个可选择的认证数据域。在 Windows 中, $cname$ 保存有主角名字和领域 (例如 $diego@tuhh.de$), 认证数据保存有组 SID。Microsoft 公司实现 Kerberos 的细节可以在 *Brown(2002)* 的著作中找到。

12.4.3 委派

在分布式系统中, 顾名思义, 受控调用有了新含义。用户可以在本地节点登录, 然后在远程节点上执行一个程序。为了获得对远程节点上资源的访问, 执行这个程序将需要相关的访问权限。在典型情况下, 程序将被赋予用户的访问权限, 然后以这个权限在远程节点上运行。这个过程称为委派 (delegation)。

在远程节点上的程序, 现在以用户委派的所有访问权限运行。在一个分布式系统中, 用户可能不满意把它所有的权限交给一个没有什么控制权的节点。如果远程节点上存在弱保护, 攻击者就可能攫取用户的访问权限, 并且为了一些非法的目的使用这些权限。因此, 你可能更喜欢这样的系统: 在系统中用户自己可以控制究竟委派哪些的权限, 以及能够有问责性机制来监控委派的访问权限的使用。

Kerberos 在 Windows 中支持不同模式的委派。当解释这些模型的时候, 我们可沿袭惯例, 称参与者为 Alice 和 Bob。所以, 当 Alice 需要 Bob 提供服务时, Bob 得有对她服务的访问控制权

限, 当她预先知道 Bob 需要什么的时候, 她会为相关的服务申请代理票据 (proxy ticket), 并且把这些票据和相应的会话密钥交给 Bob。这些服务票据标注为代理票据, 而且应该包含限制 Bob 怎样使用 Alice 信任的专门认证, 例如, 说明允许 Bob 打印文件名。如果 Alice 预先不知道 Bob 将需要什么, 她可为 Bob 申请一个前向 (forwarded) TGT, 并且将这张票据和相应的会话密钥传送给 Bob。Alice 将她的身份委派给 Bob。Bob 现在便可以以她的名义申请票据。Brown (2000) 著作中称这一“快速、松散的方式为委派信任”。主体被命名为 OK-AS-DELEGATE 来管理信任的委派。

委派身份是个好主意吗? 在主角的这一层是相当合理的。SID 给予访问控制权限, 代表用户运行的进程不能频繁地回访用户以及询问将 SID 赋予另一个进程是否合适。然而, 当一个用户 Alice 将她的身份委派给另一个用户时, 就相当于已将她的密码泄漏出去 (因为此时转交的票据是有效的)。共享密码通常是不允许的, 并且违反了 Alice 的共同安全策略。

来自操作系统安全的概念可能在应用层并不总是那么适当。调用两个用户和进程 Alice 和进程 Bob 都不会增加透明度。一般来说, 类似“Alice 与 Bob 会谈”或“Bob 核实 Alice”的拟喻很容易引起误导。在计算机安全中, 实体是计算机, 而不是人的活动, 它们通过网络发送消息, 它们并不进行交流, 你必须决定核实 A 的身份意味着什么。很明显在计算机之间没有视觉接触。

经验教训

Alice 和 Bob 是一个美丽的陷阱。他们允许你讲述美好的故事, 但他们也会误导你。

12.4.4 撤销

如何撤销主角的访问权限呢? KAS 和 TGS 的系统管理员必须更新他们的数据库, 以便访问权限对主角来说不再有效。这个访问权限对下一次的会话, 即这个主角再次登录系统或者从 TGS 请求一张票据时无效。而这个主角已经拥有的票据在期满以前都有效。例如, KAS 票据通常有大约一天的生存期。这是 TOCTTOU 问题的另外一个例子。

现在, 你不得不面临一个在方便和安全两个方面找寻平衡的问题。如果 TGS 给出了一张失效期很晚的票据, 主角则不需要频繁访问 TGS。同时, TGS 偶尔离线对用户方面也没有什么影响。然而, 访问权限的撤销将有一个较长的延时影响。相反如果 TGS 提供的票据具有一个较短的生命期, 主角必须更为经常地更新它们的票据, 安全服务器的可用性对于系统性能来说也变得越来越重要了。

12.4.5 小结

为了全面评定 Kerberos, 你就必须跳出分析认证协议和基础加密算法的强度圈子。必须还要考察它的非密码安全特性。下面列出了需要考察的部分要点。

- 消息的及时性通过检查时间戳来确认。因此, 在整个系统中要求合理的时钟同步, 而且时钟自身必须加以保护, 以防止攻击。安全时钟同步自身也要求认证。
- 时间戳检查允许一些时钟有偏差。典型的五分钟接收窗口相当大, 且比较容易遭受重放攻击。
- 服务器必须在线。KAS 需要在线登录; 请求票据时也需要 TGS。如在上面讨论的, TGS 这种可使用性的要求可能放宽。
- 会话密钥 (用于对称加密器) 由 Kerberos 服务器 (认证服务器和票据授予服务器) 生成。当这个会话密钥在主角之间的后继通信中使用, 对服务器的信任必须包含以下信任: 服务器将不会滥用它们的能力去窃听。

- 口令猜测和口令欺骗攻击是可能的(Wu, 1999)。
- 密钥和票据保存在客户的机器上。因此,你依赖于这个节点上的保护机制去维护 Kerberos 的安全性。只要 Kerberos 用户工作在简单终端上,这就不是个大问题。一旦用户在 PC 机或者在多用户工作站上运行 Kerberos,那么情况就发生了改变。
- 初始化的客户要求是不会被认证的。一个攻击者发送一个欺骗性的认证要求将得到票据。这可以构成一个对认证服务器的拒绝服务攻击,或者算是一个为密码破译攻击收集材料的企图。作为对策,服务器可以要求用户在产生任何票据之前的每个阶段均进行认证。

此外,将 Kerberos 执行的安全性中与协议自身的安全性区分开来是非常重要的。比如,据说 Kerberos 第 4 版用一个弱的随机数生成器来产生密钥,这样就可以很容易通过暴力搜索找到密钥。

12.5 公钥基础设施

STS 协议的描述忽略了一个很重要的细节。A 和 B 怎么知道它们用来检查签名的验证密钥确实与正确的参与者相符合。这是一个在公共密码学中至关重要的问题。我们很少在密钥之间运行协议,而是运行于那些在更高的协议层有实义名字(身份)的主体之间,例如,客户的名字或服务器域名系统的名字。这里必须有一个可信任的连接身份和密钥的资源。注意,有了对称加密器,参与者 A 和 B 常常信赖服务器来创建这个连接。

12.5.1 证书

Diffie and Hellman(1976)设想,有这样一个公共地址目录,在那里你可以查询用户的公共密码,就像在电话簿里一样。1978 年,在一个学生的项目中, Kohnfelder 将这样一个目录用作一组数字签名的数据记录,记录中含有名字和公钥。他将这些记录命名为证书(certificate)。证书最初都有单个的函数,并绑定名字和密钥。今天,这个术语被更广泛地应用开来,你可以看到像下面这样的解释。

证书:授权给主体的有符号的工具。它至少包括一个发行者和一个主体。它包含有效的条件、认证和委派信息(Ellison et al., 1999)。

依照这种思想,我们可以将证书定义为数字签名文档,该文档将一个主体绑定到一些其他信息上。主体可以是人、密钥、名字等。身份(ID)证书绑定名字和密钥。有时这仍然是对术语证书的默认解释。属性证书绑定名字与授权认证。授权证书绑定密钥与授权认证。

12.5.2 证书权威

在主体与密钥之间绑定,或是与一些其他信息的绑定,是由发行(签署)证书的参与者建立的。这个参与者就被认为是发行者。证书权威(Certification Authority, CA)是发行者的另一个名字。当你为个人使用而发行证书的时候,你就是一个 CA。有时,CA 仅指发行 ID 证书的组织。应用决定了 CA 必须满足的技术上和程序上的信任需求。CA 必须保护它自己的私钥。它可能必须核实他(她)所声称的主体和证书中的属性是否正确。检查可能在完全不同的级别上进行。VeriSign 证书级别就是这样的一个例子。对大多数基本级别,CA 只需检查主体的名字是否唯一。对最高级别,主体则必须带着政府认可的身份文件亲自现身。这些检查中的一部分可能由注册权威(Registration Authority, RA)执行,而且 CA 的主要任务是发行证书。

用公钥基础设施(Public Key Infrastructure, PKI)描述发行和管理证书的体系不怎么严密。不同资源决定不同的 PKI,它有可能是:

- 管理数字证书的软件。

- 集提供安全保障的硬件、软件、策略和人为一体的系统。
- 保障网络安全的技术。
- 数字 ID 卡的全球系统。

这里没有进行精确的定义。任何时候你遇到一个所谓的 PKI，在进一步下结论之前，你必须建立起适合该特定场合使用的 PKI 解释。

12.5.3 X.509/PKIX 证书

今天，X.509 版本 3 是采用 PKI 标准的最广泛的一个版本。最初的 ITUT 推荐版本 X.509(国际标准化组织，1997)是 X.500 目录的一部分(CCITT，1998)，该目录随后被 IS 9594-1 采用。X.500 被打算作为一个全球的，已命名的实体(如人、计算机、打印机等)的分布式数据库，也就是全球在线的电话本。X.509 证书将公钥(最初的口令)绑定到 X.500 的路径名(区分的名字)来注释谁有修改 X.500 地址目录节点的许可。X.500 与基于身份的连接控制权相吻合。

事实上，所有这些安全服务都依赖于通信间彼此知道的可靠的参与者的身份，即认证(国际标准化组织，1997)。

在这个超前于 applet 程序和许多新的电子商务方案的时代，一种不同的存取控制会更适当。

比起先前的版本，X.509 v3 证书的形式(如图 12-5)包含了一些扩展来增加适应性。扩展被标记为临界。如果一个临界的扩展不能被一个执行程序执行，这个证书必须被拒绝。非临界的扩展可能被忽略。临界扩展可以被用于规范证书的使用策略。

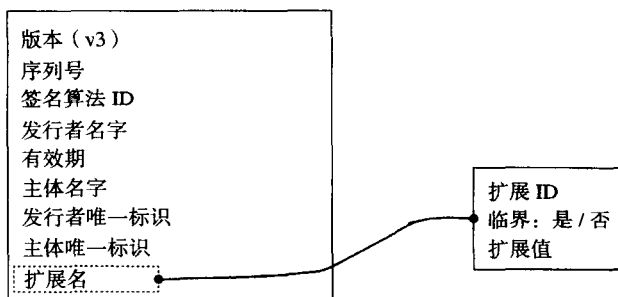


图 12-5 X509.3 v3 证书格式

PKIX 是因特网 X.509 PKI(Housley et al., 2002)。它指定适当的扩展来使 X.509 v3 来适应因特网。公钥证书(public key certificate, PKC)包含主体的公钥和一些其他的信息。属性证书(AC)包含对主体的属性集。属性证书由属性权威(Attribute Authority)来发行。PKI 是硬件，软件，人员，策略，需要被创建、管理、存储，分发的程序和撤销 PKC 的策略和程序的设置。特权管理基础设施(Privilege Management Infrastructure, PMI)是 AC，和他们所发行的属性权威、主体、可信赖的参与者及知识库的集合。

X.509 和 PKIX 是以名字为中心的 PKI。为了将密钥和访问控制权相连接，你必须知道密钥属于哪个主体。授权属性通过公钥证书和属性证书中的一个共同的名字来连接密文密钥。另一个可选择的是 SPKI，简单公钥基础设施(Simple Public Key Infrastructure; Ellison et al., 1999)。SPKI 是以密钥为中心的 PKI。SPKI 证书直接绑定密钥与属性(访问控制权)。

经验教训

像用户身份一样，证书用于两个目的。它们能够识别与密文密钥相连接的实体，或者它们能够指定被授予密文密钥持有者的访问控制权限(在没有揭露持有者身份的情况下)。

12.5.4 证书链

从技术上来讲,需要证书核实数字签名的说法是不正确的。正确的是需要验证密钥的认证备份。验证密钥可以存储在证书中,但也可以存储在受保护的内存中。然而,也有系统要求所有验证密钥均存储在证书中的。Java 2 的安全模型就是一个例子。

为了检查某个证书,需要另一个验证密钥,这个验证密钥由一个证书担保。这就创建了一个证书链。最后,你需要一个根验证密钥(root verification key),该验证密钥的真实性不能由一个证书保证。典型的就是在浏览器或邮件程序中安装的一组根验证密钥。当你接收到类似“你不信任这个证书”这样的消息,那么在那些根验证密钥中就没有证书链。在一个证书需要签名确认的系统中,你可以用自己签署的证书来存储根密钥。一个自己签署的密钥可以用它自身包含的公钥来验证。

证书有一定的期限。认为证书过期后就不能再使用是不正确的。决定该怎样处理过期的证书是一个策略决策。就拿护照来说,一个欧盟国家护照在它有效期过期一年后,在欧盟国家旅行仍然是有效的。当评价一个证书链的时候,主要有两项策略适用于期满日期(或吊销)的情况。

在 shell 模型中,所有的证书在评估的时候都必须有效。如果一个高级别的证书期满或者被吊销,所有被其相应的签署的证书,其相应的私钥必须用一个新的密钥重新发行。这个老方法在 SP-KI 中执行。当采用 shell 模型的时候,一个 CA 应该只在它自己的证书期满之前发行证书。

在链模型中,发行者的证书必须在证书发行的时候有效。如果一个高级别的证书期满或是被吊销,所有被私钥签署的证书,其相应的私钥仍然有效。这个模型要求在证书发行的时候,建立一个可靠的时间戳服务。一个时间戳认证(Time Stamp Authority, TSA)就是一个 TTP, TTP 及时地为一个特殊数据的瞬时存在提供了存在的证据。TSA 不会检查它所证明的文件。TSP, PKIX 的时间戳协议在 RFC3161 中有描述。

12.5.5 撤销

如果证书相应的私钥已经泄露或是如果那个证书的担保已失效,那么这个证书可能不得不被撤销。对此, X.509 中提出的方法为:证书撤销清单(Certification Revocation List)按有规律的间隔分布或者按需分布。如果在线检查不可能或者太昂贵,那么 CRL 就显得有意义了。如果在线核实可行,则可以在线询问 CRL。但是当在线核实可行的时候,可以在线询问证书身份,此时 CRL 可能就变得多余了。相反,证书的当前身份由像在线证书身份协议(OCSP, RFC2560)这样的协议进行查询。举例来说,德国签名基础设施需要检查有效证书的实际清单。对于撤销来说,短暂证书是一个选择。

12.5.6 电子签名

手写签名在商业和法律事务办理中扮演着很重要的角色。(不是因为它们不容易被伪造,而是因为它们发送信号意图。)当用电子方式执行这些事务办理的时候,就需要一个与此相同的手写签名。数字签名作为一种解决方法被提出来。然而,数字签名仅仅是用于连接文件和个人的安全服务。连接文档和个人的电子签名服务通常被称作电子签名(electronic signature)。电子签名常常用数字签名作为一个构建块,但没有它们也可以执行。

人们为了将电子签名结合到法律系统中已经做了不少努力。一个显著的例子就是欧盟电子签名指示(1999 年 12 月 13 日关于电子签名的共同体框架的 1999/93/EC 指令)。这个指令将电子签名作为一个技术核心词,但是所谓先进的电子签名事实上需要和数字签名一起执行。先进的

电子签名由合格的证书担保。对认证服务商(CA 等)和签字制造设备(例如智能卡)的进一步的要求出台了。图 12-6 给出了一个安全电子签名服务及其安全基础的组件。

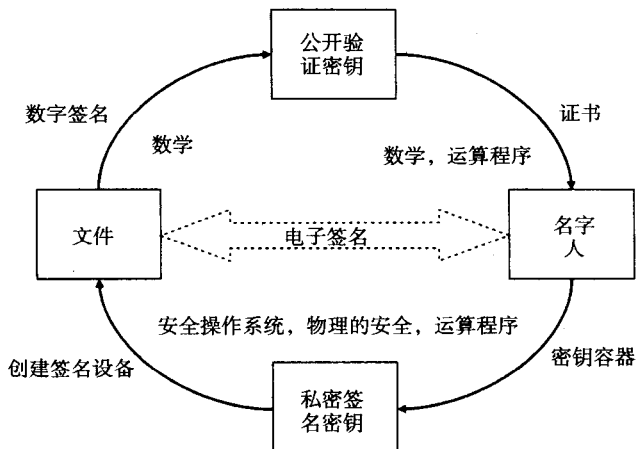


图 12-6 用数字签名进行电子签名服务的组件

经验教训

最后，密码保护不得不在一个非密码学的基础上。

12.6 可信计算—证明

让我们把重点从人转到机器。如果我们确切知道安装了硬件和软件的哪些配置，那可能在某些情况下，我们只通过远程平台就能从事交易。我们可能会从远程平台上请求获取一些信息，但怎么才能确定获得的信息是正确的呢？这是可信计算组(TCG)提出的其中一个问题。对信息准确性的担保过程称为证明(attestation)。

所谓的可信平台模块(Trusted Platform Module, TPM)是在 TCG 中专门定义的安全体系核心中的硬件组件。在 TPM 中，硬件和软件的完整性检测值存放在平台配置寄存器(Platform Configuration Register, PCR)中。此外，TPM 还包含了一个内嵌的背书密钥(Endorsement Key, EK)，这个密钥是不能被移除的。EK 是一个 2048 比特的 RSA 密钥对，其中公钥用于识别 TPM 和私钥用来解密被传送到 TPM 的消息。应该能够确定一个 EK 是否属于真正的 TPM。一般来说，这由 TPM 厂商的证书实现。

TPM 能够用一个信任密钥来签署 PCR 的内容，但是如果用 TPM 发行的证明总是用同样的密钥签署，观测者可以把它们都连接起来。为了使证明不可连接，TPM 因此创建了证明身份密钥(Attestation Identity Key, AIK)。AIK 是由 TPM 产生的 RSA 签名密钥对。TPM 需要一个 TTP 的服务，也就是所谓的私密 CA(pCA)，来获得一个证书确定 AIK 是否属于真正的 TPM 证书。下面的协议曾被考虑用来获得这样的证书。TPM 将它的公共 EK 和证明身份密钥 AIK_i 的公共部分发送到 pCA。CA 核实 EK 是否属于真正的 TPM，在 EK 和 AIK 之间存储映射，并且将证书 Cert_{pCA} 返回到 TPM。证明中，当 TPM 在一个证明中用 AIK_i 的私密部分来签署 PCR 的内容时，它包含了发送给验证人信息中的 Cert_{pCA}。

- (1) TPM → pCA: EK, AIK_i
- (2) pCA → TPM: Cert_{pCA}
- (3) TPM → Verifier: AIK_i, sAIK_i(PCR), Cert_{pCA}

你可能注意到了企图使证明不可连接的行为失败了。在第一个消息中所有的证明密钥都连接到了 EK, 这样, 所有的证明仍然能被连接。这个问题已经得到了进一步的解决, 例如, 直接匿名证明 (Direct Anonymous Attestation)。然而, 完全匿名不理想, 因为还有规定称它应该可能确认来自己知的已经失密的 TMP 证明。协议说明了这些竞争目标的提出方式, 例如, 在 *Brickell, Camenisch and Chen(2004)* 的著作中, 以及 *Camenisch(2004)* 的著作中。

12.7 深层阅读

Kerberos 的详细说明包含在 Internet RFC 1510 (*Kohl and Neumann, 1993*) 中。与开发环境内容放置在一起的 Kerberos 安全分析在 *Bellovin and Merritt(1990)* 的著作中有所论述。Kerberos 扩展完善了它的接入控制功能, 例如, 通过 SESAME (*Ashley and Vandenwauver, 1999*), OSF DCE 或者 PERMIS 中的特权属性证书 (Privilege Attribute Certificate, PAC), 与 Kerberos 相似的是, KryptoKnight 是一个集中系统, 在这个系统中安全服务器提供认证和密钥分发。通过用完整性检测代替加密来绕过 20 世纪 90 年代遇到的输出问题。

先不管它们的短暂性, PKICX roadmap (*Aressenault and Turner, 2000*) 和 SPKI 的因特网草案 (*Ellison et al., 1999*) 应该在证书理论的讨论中有所提及。在一个国际公司中配置 PKI 得到的经验教训在 *El-Asa et al. (2002)* 的著作中有所报道。

12.8 练习

练习 12.1 在 http 基础认证协议中, 当客户端发送一个密码的哈希函数而不是一个 base64 编码的密码时, 分析安全收益 (如果有的话)。

练习 12.2 阐述 AKEP2 协议提供的双向实体认证和隐式密钥认证。

练习 12.3 阐述 STS 协议提供了双向实体认证和显式密钥认证。

练习 12.4 考虑运行在一个用户 A 和服务器 B 之间简单的基于口令的质询 - 应答协议。 P_A 表示了 A 的密码, n 是由服务器产生的随机序列号, 并且 h 是一个已知的加密哈希函数。

$$\begin{aligned} (1) S \rightarrow A: & \quad eP_A(n) \\ (2) A \rightarrow S: & \quad eP_A(h(n)) \end{aligned}$$

试说明该协议很容易遭受离线密码猜测攻击。

练习 12.5 试证明 EKE 协议是不容易遭受离线密码猜测攻击的。

练习 12.6 试证明 STS 协议不会像 Diffie-Hellman 协议一样受到 man-in-the-middle 攻击。

练习 12.7 试修改 Needham-Schroeder 密钥交换协议, 使参与双方 A 和 B 都能通过输入帮助会话密钥的产生。

练习 12.8 对 KryptoKnight 协议作一个简短的描述, 并且讨论对比 Kerberos 来说, 它有哪些优点和缺点。

练习 12.9 设计一个 Kerberos 的扩展实例以便允许在域间进行访问。为了使该方案可行, 需要哪些管理安排, 该协议中需要增加哪些步骤?

练习 12.10 公司会保存已签署的电子记录的档案文件, 还有相关的证书。当在档案文件中检查文档的有效性时, 应该使用 shell 模型或是链模型?

练习 12.11 考虑一项方案, 方案中的证书用来委派访问控制权。在决定访问控制的要求时, 应该使用 shell 模型或是链模型?

练习 12.12 考虑一项方案, 方案中的证书用来委派访问控制权。主体应该能够授予不能行使自己的权利吗? 在回答中, 请区分以名字为中心和以密钥为中心的公钥基础设施。

第13章 网络安全

网络连接着计算机。这种连接祸福参半。更多的交互成为可能，但同时也有更多不受欢迎的交互。你可能想控制网络上的用户怎样能访问你的系统，自己的系统上的用户怎样能访问网络以及怎样保护你的数据在网上传输。因此，网络安全不仅仅是应用密码学，还包括网络级访问控制和入侵检测。

目标

- 了解安全问题对网络的挑战，理解网络安全对计算机安全的影响和依赖。
- 以基本的 Internet 安全协议 IPsec 和 SSL/TLS 为范例，介绍网络安全协议的设计。
- 了解网络边界怎样用作安全周界。
- 理解防火墙和入侵检测系统的原理和局限。

13.1 引言

计算机网络是分布式系统中节点之间传输数据通信的基础设施。一个节点上的应用程序将要发送的数据事先准备好，用一个电信号或光信号序列传送出去，接收端进行重组，并发给应用程序。网络协议必须找到一条从发送者到接收者的路径，必须处理数据的丢失和损坏以及失去的连接，例如施工人员切断了电话线缆。一个好的工程实践方法是逐一解决这些问题，并使用分层体系结构，把应用协议放在顶部，把有关物理传输的编码信息放到底部。

ISO/OSI 安全结构(国际标准化组织, 1989)定义安全服务(security service)来阻止网络安全威胁。安全服务由安全机制(security mechanism)来执行。提供的这些服务的机制大部分来自密码学。典型的如，加密、数字签名、完整性检查功能等。密码保护有一个好的特性：当 N 层安全协议在其下层不安全的协议的上面运行时， N 层的安全协议不会受到危害。对这种特性有一种例外。当你的目标是匿名，而且你采取了预防措施来隐藏在某一层的参与者的身份时，则由较低层的协议添加的一些数据仍然可能泄露关于消息的来源和目的地的信息。

不论怎样，并非所有的安全问题都能用密码学解决。进一步的防御是访问控制机制(防火墙)和入侵检测系统。设计协议对付拒绝服务攻击或限制这种攻击不放大是网络安全中的另一个令人关注的问题。

13.1.1 威胁模型

网络上的攻击者可以是被动的或主动的。被动攻击者仅仅监听通信。当攻击者仅仅是监听通信时，我们讨论偷听(eavesdropping)、搭线窃听(wiretapping)或嗅探(sniffing)。当攻击者无法读取私人消息时，通信量分析可以试图确定通信参数。攻击者会试图识别相同的源(链接)或者找出谁在和谁通信，以及其通信频率。在移动服务中，攻击者也许也对用户的位置感兴趣。

主动攻击者也许会修改消息、插入新消息或者毁坏网络管理信息，如在域名系统(DNS)中的名字和 IP 地址。在欺骗攻击中伪造消息发送者的地址。在洪泛攻击中，大量的消息被指向受害者。在蹲守攻击(squatting attack)中，攻击者要求定位受害者。主动攻击并非比被动攻击更加难于实施。事实上发一封伪造发送人的邮件比拦截别人的邮件容易多了。

攻击者会试着了解你的网络内部结构, 并使用这种信息来发动攻击。来自于网络管理协议的信息就是安全敏感的, 因为这些协议收集的是有关节点的负荷以及可用性的诊断信息。同时, 这些协议需要高效地使用网络。如果过分地保护这些协议, 网络提供的服务质量会大大地降低。

13.1.2 通信模型

在第 12 章中, 安全协议在抽象层中描述。消息直接在主角之间传递, 我们也没有考虑这种交换的精确性。安全协议的分析常常在这种模型中进行。Internet 用云朵形状表示。信息可以被任何决心要发送或修改的人发送或修改。攻击者可以通过协议分析在正常的网络模型中控制所有的信息流动。用笔和纸来做比喻就是, 发送者在一张纸上写好信息, 把它扔到地板上。之后, 接受者到他或她的垃圾箱看有什么东西没有。

这样一个抽象模型并非一个最好的能解决所有安全问题的模型。在安全分析中, 我们可以假定有一个不是很强的潜在真实对手。比如, 我们可以假定对手不能控制整个网络, 他仅仅只能读那些直接发向他的地址的信息。但是他可以随心所欲地修改发送者的地址。在 13.1.3 节中, 给了一个这种情况的安全分析的例子。在设计协议时, 我们可能发现网络中的实体, 比如防火墙或 NAT, 会妨碍 Alice 和 Bob 运行他们的协议, 需要特别地注意。

13.1.3 TCP 会话劫持

与客户机服务器 B 创建一个 TCP 会话, 客户机 A 需要先启动下面三步握手协议。

- (1) $A \rightarrow B$: SYN, ISSa
- (2) $B \rightarrow A$: SYN | ACK, ISSb, ACK(ISSa)
- (3) $A \rightarrow B$: ACK, ACK(ISSb)

SYN 和 ACK 表明各自的位已经被设置好了。ISSa 和 ISSb 是 32 位序列号。它们的关系是 $ACK(ISSa) = ISSa + 1$ 和 $ACK(ISSb) = ISSb + 1$ 。

假设消息总是发送到指定的接收者, 而且在传输过程中不能被观察到。攻击者只能插入假的源地址和他自己的消息。只要序列号是随机的, 这个协议就是安全的。攻击者就不得不猜发送到另一方的序列号。因此, RFC 793 指定每 4 微秒让 32 位的计数器低位增加 1。但是, Berkeley 导出 Unix 内核每秒增加 128, 每个新连接增加 64。因此, 就没有太多的随机性来混淆攻击者了。

早在 1985 年就有利用这种执行决策的攻击 (Morris, 1985), 并且在 1989 年被推广 (Bellovin, 1989)。攻击者 C 先和他的目标 B 开始一个真实的连接, 同时收到一个序列号 ISSb。然后, 攻击者假扮 A , 发送一个在源地址填入了 A 的地址的包。

$C(A) \rightarrow B$: SYN, ISSc

B 回应

$B \rightarrow A$: SYN | ACK, ISSb', ACK(ISSc)

到合法的 A 。 C 不能看到这条消息, 但是他使用 ISSb 来预测当前 ISSb', 发送

$C(A) \rightarrow B$: ACK, ACK(ISSb')

如果猜测是正确的, B 将和 A 连接, 尽管是 C 在发包。 C 不能看到这个会话的输出, 但是它也许可以使用 A 在 B 上的权限执行命令。这种攻击方法可以在 Unix 环境下运行, 在此环境下攻击者可从可信主机 A 处骗取消息 (参见 6.7.2 节)。像 rsh 这样使用基于地址验证的协议, 假设从一个可信主机登录的用户就是被验证了的, 这将是非常脆弱的。

让防火墙封锁所有来自于局部来源地址的所有 TCP 包, 就可以防御这种攻击了。这种方式只能在所有你信任的主机都位于一个局域网中时, 才起作用。如果还有在 Internet 中受信任的主

机, 防火墙就不得不阻塞所有使用 TCP 的协议和基于地址的认证。一个更好的解决方案是, 完全避开基于地址的认证。密码认证更好一些。

13.1.4 TCP-SYN 洪泛攻击

在应答了第一个 SYN 包之后, B 服务器会存储 ISSb 序列号, 来验证客户端发的 ACK。攻击者在使用 TCP SYN 洪泛攻击的时候, 初始化大量的 TCP 请求 (SYN 包), 但是并不完成协议的运行, 直到 B 服务器达到了它的半开连接的极限, 再也不能应答新的请求为止。将修改 TCP 握手协议允许服务器动态丢弃保留的状态。作为 TCP 会话劫持攻击的一部分, C 能够发动 SYN 洪泛攻击 A , 因此 A 不能处理来至 B 的 SYN-ACK 包, 也不能丢掉攻击者想打开的连接。

13.2 协议设计原则

ISO/OSI 结构的七层模型 (图 13-1) 是分层网络协议的常见框架。分层模型对讨论网络安全提供了相当有用的抽象概念。分层模型也让我们回到了 2.4.2 节中的熟悉主题。在顶部的安全服务可以用来满足特定应用的要求。然而, 不同的应用都需要他们自身的安全协议。底部的安全服务可以用来保护所有的高层通信, 减轻应用层协议设计者对安全的关心。然而, 一些应用可能发现这些保护不能很好地满足它们的要求。

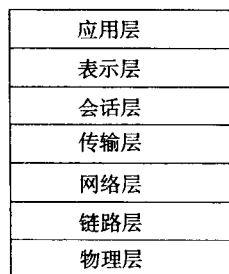


图 13-1 ISO/OSI 七层模型

在分层模型中, N 层的对等实体 (peer entity) 使用 N 层协议。 $N+1$ 层协议看到的是在 N 层上的一个虚拟连接, 并且不会考虑任何底层的情况 (图 13-2)。当然, N 层协议是建立在低层协议之上的。这里有一个用于传输数据到底层的通用模式。 N 层协议的消息称为 N 层协议数据单元 (protocol data unit, PDU)。 N 层协议通过调用 $N-1$ 层的设施来传输一个 N 层的协议数据单元。在这个阶段中, N 层协议数据单元可能被分段或者经过其他的处理。再对结果增加头部和尾部使其成为 $N-1$ 层协议数据单元 ($(N-1)$ -PDU)。这些 $N-1$ 层协议数据单元的接收者使用报头和报尾中的信息来重组 N 层协议数据单元 ((N) -PDU)。图 13-3 给出了这个过程的简化视图。

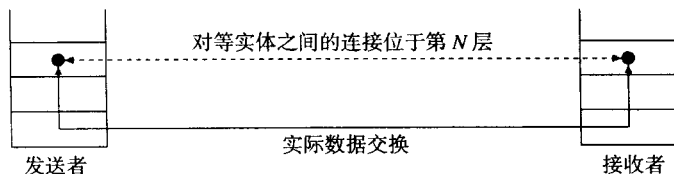


图 13-2 在 N 层的虚拟连接

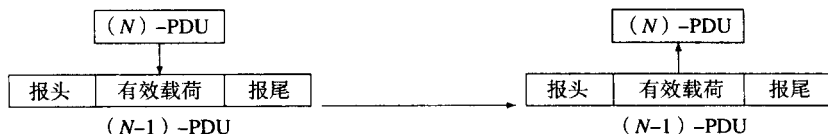


图 13-3 处理一个 (N) -PDU

有两个重要的选择来实施 N 层协议调用的 $N-1$ 层的安全服务。较高层的协议能够察觉更低层的安全协议, 或者安全协议是透明的。在第一种情况下, 高层的协议不得不改变它的调用来访问提供的安全设施。在第二种情况下, 高层协议根本不用改变。在两种情况中, $N-1$ 层协议数

据单元的帧头是一个比较方便存储安全相关数据的地方。

网络协议栈有四层(图 13-4)。在应用层的协议有 telnet、ftp、http、smtp(Simple Mail Transfer Protocol, 简单邮件传输协议)或者 SET(Secure Electronic Transaction) 这些协议。在传输层有 TCP(Transmission Control Protocol, 传输控制协议)和 UDP(User Datagram Protocol, 用户数据报协议)这些协议。在网路层有 IP 协议。TCP 和 UDP 使用端口号来标识协议数据单元属于谁。常用的端口号是 21(ftp), 23(telnet), 25(smtp, 发送邮件), 110(pop3, 接收邮件), 143(imap, 接收邮件), 80(http), 443(https, 安全网页)及 53(DNS, 名字查找)。链接(和物理)层的协议是使用的特殊的网络技术。

应用层
传输层
国际互联层
链接层

图 13-4 Internet 分层结构

TCP 和 IP 以及 UDP 和管理协议 ICMP 是网络的核心。最初, 这些协议是设计来使友好和合作的用户能通过可靠的网络连接的, 因此完全没有关心过安全事宜。今天, TCP/IP 被广泛地使用, 并要求更强大的安全性。因特网工程任务组(Internet Engineering Task Force, IETF)已经对因特网安全协议和请求注释(Requests for Comments, RFCs)中的进行了标准化。IETF 对为数众多的使用中的安全协议进行了修订, 并发布了新的网络安全协议。

13.3 IP 安全

IP 是一个无连接无状态的传输 IP 包的协议。这些 IP 包就是网络层的协议数据单元。核心 IP 规范提供尽力的服务。IP 是无连接无状态的, 因此每个数据报被视为一个不依赖于其他任何 IP 数据块的独立实体。没有什么能保证包能投递到目的地, 没有机制来维持包的顺序, 也没有安全协议。IPv4 作为 RFC 791 在 1981 年发布。之后, 因特网不断发展, 结果 IP 协议不得不适应新的要求。IPv6 在 RFC1883 中被明确提出。当讨论 IP 安全机制的时候, 我们将参考这个版本。

IP 的安全结构(IPsec)在 RFC2401 中有介绍。IPsec 在 IPv4 中是可选的, 但在 IPv6 中却是强制要求的。IPsec 包含两个主要的安全机制, 在 RFC 2402 中描述的 IP 认证报头(Authentication Header, AH)和在 RFC 2406 中讲述的 IP 封装安全有效载荷(Encapsulating Security Payload, ESP)。IP 安全架构不包括阻止通信量分析的机制。

13.3.1 认证报头

IP 认证报头只保护 IP 包的完整性和可靠性, 但是不保护机密性。最初是由于政治上的原因推荐使用它的。20 世纪 90 年代, 加密算法的输出限制为只认证机制创建了机会。如今, 这些输出限制基本上被消除, 因而推荐只使用 ESP 来简化 IPsec 的执行。

13.3.2 封装安全有效载荷

ESP 提供机密性, 数据源认证、数据完整性、一些重放保护以及限制传输流的机密性。ESP 包(图 13-5)包含下面这些域。

- 安全参数索引(Security Parameters Index, SPI)是一个 32 位的域, 唯一用于标识数据块与目标 IP 地址和安全协议的安全结合体。
- 序列号是个无符号 32 位域, 包含一个计数值; 这个值必须被发送者计入, 而由接收者作判断处理。
- 有效载荷数据是一个包含传输层 PDU 的可变长度域。
- 填充区是一个可选的域, 它包含为加密算法填充的数据; 数据的长度被填充为算法块的

整数倍长。

- 填充长度。
- 下一个报头是传输层 PDU 的类型。
- 认证数据以 32 位字长为单位，长度可变。包含一个完整性校验码 (Integrity Check Value, ICV)，由 ESP 包减去认证数据 (Authentication Data) 计算得到。

SPI 和序列号构成了 ESP 的报头。在有效载荷后的域是 ESP 的报尾。ESP 可以使用两种模式。在传输模式中 (图 13-6)，上层协议框架，例如来自 TCP 或者 UDP 的框架，被封装在 ESP 中。而 IP 报头并未加密。传输模式为两个终端主机间交换的包提供端到端的保护。当然，两个节点都必须支持 IPsec。

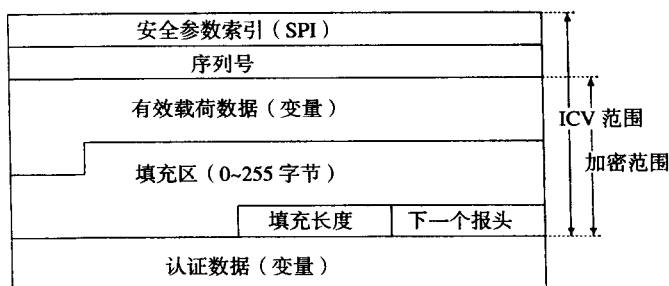


图 13-5 ESP 包

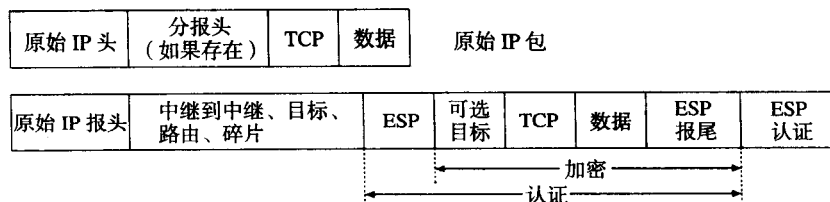


图 13-6 传输模式下对 IPv6 包应用 ESP

在隧道模式 (tunnel mode) 中 (图 13-7)，完整的 IP 数据报与安全域一起被视为外部 IP 数据报的新的有效载荷。原始内部 IP 被封装到外部 IP 数据报。IP 隧道因此可以描述为 IP 中的 IP。隧道模式可以用在 IPsec 已经在终端主机的网关上实现的情况下。终端主机不需要支持 IPsec。网关可以是一个边界防火墙或者路由器。这种模式提供网关到网关的安全，而非端到端的安全。另一方面，我们使通信流保持了机密性，使得中间路由对于 IP 数据报不可见，并且原始来源和目标地址得到了隐藏。

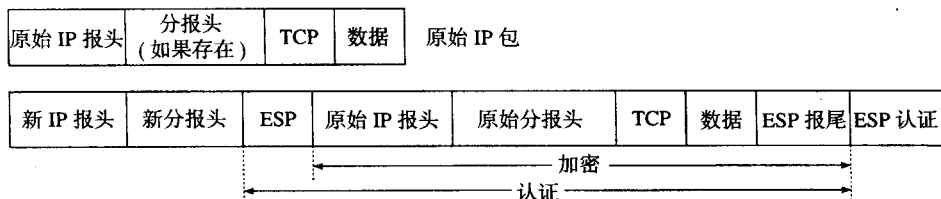


图 13-7 隧道模式下将应用 ESP 到 IPv6 包

13.3.3 安全关联

通常, 破解或者验证一个 ESP 包的系统必须知道使用的算法和密钥(以及初始化矢量)。这种信息都存储在一个安全关联(Security Association, SA; RFC 2401)中。安全关联通常是在两个通信主机之间建立的单向的逻辑连接; 定义了双方如何使用安全服务进行通信。两台主机之间的双向通信需要两个安全关联, 每一个 SA 保护一个方向的数据。因此, 安全关联通常是成对创建的。

一个安全关联是由 SPI(AH 和 ESP 报头携带), IP 目的地地址以及安全协议(AH 或 ESP)标识符唯一定义。它还包含像算法标识、密钥、密钥时限和可能的 IV 等相关联的密码数据。还有一个序列号计数器和一个反应答窗口。SA 也表明是工作在隧道模式还是传输模式。活动 SA 表保存在 SA 数据库(SAD)中。SA 还能够合并起来, 例如, IPsec 隧道的多层次嵌套。每个隧道可以在不同路由的 IPsec 网关开始和结束。图 13-8 所示的便是一台带着安全关联的网关和内部主机的远程主机的一种典型配置。

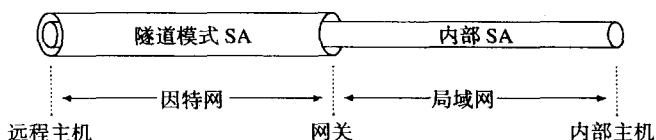


图 13-8 SA 组合

13.3.4 因特网密钥交换协议

如果节点的数目很小, 而且没有放大到适当的支持 IP 安全协议的主机网络数量, 就可以手工创建 SA。手动加密的备选方案之一就是因特网密钥交换协议(Internet Key Exchanged Protocol, IKE; RFC 2409)。一个新的版本, IKEv2 正在制定中。它的安全目标就是实体认证和建立新的可以共享的密钥。使用共享密钥来获得进一步的密钥。其安全目标还包括所有密码算法的安全协商, 例如, 认证方法、密钥交换方法、解密算法和 MAC, 或者哈希算法。一种 cookie 机制(不要和 http 的 cookie 混淆)用来提高对拒绝服务攻击的抵抗能力。有选项来设置保密性为不完全安全或者一定是安全的。

IPsec 提供很多选项和参数来灵活地创建密钥。一些人也许会说, 太多啦。IP 安全协议需要大量的对称密钥。每个 SA 都有一个密钥, 后面的每个组合中都有很多的 SA,

$$\{\text{ESP, AH}\} \times \{\text{transport}\} \times \{\text{发送者, 接收者}\}$$

IKE 在两个阶段中起作用。阶段一, 设置一个 SA 安全通道来传输进一步的 SA 系统, 以及错误信息和传输通信量。这个阶段包含可靠的实体鉴别和密钥交换。第一阶段的协议有两种变体。一种是慢但更安全的主模式(六信息), 一种是更快的挑战模式(四信息)。主模式和挑战模式都由多种认证机制构成。基于签名和共享(对称)密钥的认证已经在实际中部署了。基于公钥加密和修改的公钥加密的认证是规定的一部分, 但并没有在实践中大量使用。阶段二, 通常使用的 SA 开始通信。通过第一阶段建立的安全通道快速通信。第一阶段允许运行许多第二阶段, 然后多对 SA 就可以进行通信了。

IPsec 尤其是认证和加密服务在创建 SA 和会话密钥时独立于密钥管理协议。因此, IPsec 的安全服务不与任何特殊的密钥管理协议相关。如果发现一种密钥管理协议有缺陷, 这个协议可以替换为其他的协议而不会影响 IP 安全协议的工具。

13.3.5 IPsec 策略

IPsec 策略决定了 IP 数据报的安全处理。支持 IPsec 的主机有一个安全策略数据库 (Security Policy Database, SPD)。SPD 会查看每个进入或出去的数据报。在 IP 数据报中的这个域和 SPD 条目的域匹配。匹配基于源和目的地址 (范围地址)、传输层协议、端口号等。一个匹配识别一个或者一组 SA (或者一个需要的新 SA)。管理 IPsec 策略和实施是很复杂的, 这是一个正在研究的领域。

13.3.6 小结

IPsec 为每个人使用 IP 均提供了传输安全保障, 而且它不改变 IP 的接口。为了实现安全, 上层协议不需要改变, 甚至不需要知道 IP 层在保护它们的通信 (图 13-9)。然而, 没有太多的范围来调整应用层要求的保护级别。我们需要关心的是 IP 作为通信协议的性能, 不能在检查应用特定的数据方面花费太多的时间来选择一种安全关联。IPsec 提供主机到主机的安全, 而不是使用者到使用者或应用程序到应用程序的安全。

由于发送者和接收者执行密码操作, IPsec 增加了协议处理开销和通信延迟时间。IPsec 可以为所有的上层协议提供安全保护, 但是它也因此增加了系统开销。IPsec 没有规定特殊的密钥管理协议。这样一来, 便可允许不同的节点选择它们最喜爱的方案, 但是在不同的节点使用 IPsec 来保护它们之间的通信之前, 它们必须就使用某种密钥管理方案达成一致。



图 13-9 IP 安全协议

13.4 SSL/TLS

TCP 协议提供在两个节点之间的可靠的字节流通信。TCP 是一种有状态的面向连接的协议, 它检测什么时候包丢失和什么时候包次序颠倒地到达, 并且丢弃重复的数据。TCP 在建立两个节点间的会话时, 可实现基于地址的实体认证。但是, 正如在 13.1.3 中强调的, 它选择了一种很容易受到攻击的执行方式。TCP 缺乏强有力的密码实体认证、数据完整性和机密性。这些服务是在安全套接字层 (SSL) 协议中引入的。它由 Netscape (网景公司) 开发, 主要保护 WWW 通信。RFC 2246 关于传输层安全协议 (Transport Layer Security Protocol) (TLS v1.0) 基本上与 SSL 的第三个版本一样。于是, 这个协议就变成了众所周知的 SSL/TLS。RFC 3268 为 TLS 定义了基于 AES 的加密算法。

在 IP 协议栈中, SSL 位于应用层协议和 TCP 协议之间 (图 13-10)。因此, SSL 能够依赖有 TCP 保证的属性, 例如它自己就不必关心数据的可靠传递。像 TCP 一样, SSL 也是有状态的和面向连接的。SSL 会话状态包括执行密码算法所要求的信息, 例如, 会话标识符、密码程序组的说明、共享的密钥、证书、由协议 (如 Diffie-Hellman 协议, 参见 12.3.2 节) 使用的随机数等。为了减少由密钥管理引起的系统开销, 一个 SSL 会话可以包括多个连接。有特色的实例是在客户和服务端之间的 http 1.0 会话, 其中为传输复合文档的每一个部分都要建立一个新的连接。对每个连接, 仅仅是状态信息的一个子集发生变化。

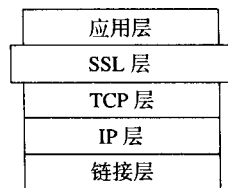


图 13-10 SSL 安全协议层

SSL 由 SSL 记录层 (Record Layer) 和 SSL 握手层 (Handshake Layer) 两部分组成。SSL 记录层从上层协议获得数据块, 然后应用在现行会话状态中的密码说明定义的密码变换。从本质上说, SSL 记录层提供一种类似与 IPsec 的服务, 在 IPsec 安全关联和 SSL 状态之间的相似绝不是偶

然的。

SSL 握手协议建立会话状态的密码参数。图 13-11 说明了在客户和服务端之间的信息交换，括号中的组件是可选的。为了举例说明这个协议，我们现在单步调试客户认证服务器的运行过程。客户用 ClientHello 消息开始协议的运行，该消息包含一个随机数、一张根据客户偏好定制的建议密码表，一个建议的压缩算法。

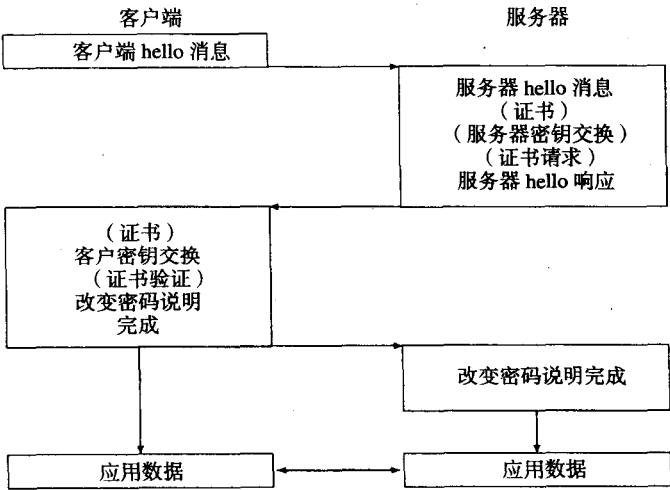


图 13-11 SSL 握手协议

M1: ClientHello:	ClientRandom[28]
	建议的密码程序组:
	TLS_RSA_WITH_IDEA_CBC_SHA TLS_RSA_WITH_3DES_EDE_CBC_SHA TLS_DH_DSS_WITH_AES_128_CBC_SHA
	建议压缩算法: 无

服务器从建议的密码组中选择 TLS_RSA_WITH_3DES_EDE_CBC_SHA。RSA 将用于密钥交换，CBC 中的三倍 DES 用作加密算法，SHA 用作 hash 函数。服务器将用一个 Server Hello 消息和一个证书链来回复：

M2: ServerHello:	ServerRandom[28]
	使用密码程序组: TLS_RSA_WITH_3DES_EDE_CBC_SHA
	Session ID: 0xa00372d4XS
证书:	subjectAltName: SuperStoreVirtualOutlet PublicKey: 0x521aa593... Issuer: SuperStoreHQ
	subjectAltName: SuperStoreHQ PublicKey: 0x9f400682... Issuer: Verisign
	Server Done: NONE

在我们的例子中，没有要求来自客户端的证书。参照证书中主体可替换名字扩展 (subject alternative name extension)，客户端验证证书链，然后本地生成一个随机的 48 字节的 PreMasterSecret。MasterSecret 是下面的前 48 字节：

PRF(PreMasterSecret, “master secret”, ClientRandom || ServerRandom)

这里，PRF 是一个基于 MD5 和 SHA 的复杂函数的简写形式。(符号 || 表示级联)。MasterSecret 用作构造如下形式的密钥块的输入：

PRF(MasterSecret, “key expansion”, ClientRandom || ServerRandom)

客户端和服务端需要的所有的 MAC 和加密密钥都可从密钥块中得到。保护从客户端到服务器通信量的密钥与保护从服务器到客户的通信量的密钥是不同的。这样，参与者可以很容易地区分他们发送的消息和接收的消息，并且他们也不受反射攻击 (reflection attack) 的影响。反射攻击就是将消息重新发送到它的发送方。

客户端现在将 PreMasterSecret 发送到服务器，它使用在选择的密码组中规定的密钥管理算法和服务器的认证公开密钥(然后，客户端应该立即销毁 PreMasterSelect)。在我们的例子中，算法是 RSA，公开密钥是 0x521aa593...。ChangeCipherSpe 消息指出随后的记录将受新商议的加密程序组和密钥保护接着客户端通过 MD5 和 SHA 构造的哈希值来捆绑第三个消息到前两个消息。

M3:	A: ClientKeyExchange:	RSA_Encrypt (ServerPublicKey, PreMasterSecret)
	B: ChangeCipherSpec:	NONE
	C: Finished	MD5 (M1 M2 M3A) SHA (M1 M2 M3A)

服务器解密 PreMasterSelect，然后从它计算出 MasterSelect、密钥块和对于客户的这次会话有效的所有衍生出来的密钥。服务器验证附加到客户消息上的 hash 值，而且用下面的消息回答：

M4:	A: ChangeCipherSpec:	NONE
	B: Finished	MD5 (M1 M2 M3A M3C) SHA (M1 M2 M3A M3C)

客户端验证在服务器消息中的哈希值。现在，参与双方都建立了它们用来保护特定应用通信的共享密钥。

小结

SSL/TLS 包含一个握手协议，客户端和服务端利用它来商定密码组，确立必需的加密资料，进行相互认证。今天，SSL 是使用的最广泛的因特网安全协议，所有的主流网页浏览器都支持它。SSL 在应用协议和 TCP 之间加上了一个安全层，因此应用程序必须明确地要求安全。这样，应用程序代码不得不改变，但是并不需要改变多少。例如，用一个 SSL 连接呼叫替换 Pre-SSL 应用程序中的 TCP 连接呼叫。SSL 连接将要初始化密码参数和原始的 TCP 连接呼叫。

客户端和服务端必需保护它们创建的安全内容参数(或者 IPsec SA)。否则，用 SSL(或 IPsec)提供的安全将受到威胁。我们再一次回到了计算机安全。

经验教训

密码保护不会受到来自在通信网络下方层的威胁，它很可能受到来自网络节点操作系统的下面层的威胁。

13.5 域名系统 DNS

应用程序常常通过 DNS (Domain Name System, 域名系统) 知道它们的组织形式。为了通信, 我们需要符合 DNS 名字的 IP 地址。这种信息由 DNS 名称服务器维护。DNS 查询给出 DNS 名称对应的 IP 地址。反向检查就是提供 IP 地址对应的 DNS 名称。攻击者会损坏这些信息来使用户错误地连到伪造的地址, 还使一些地址不能使用以及制造更严重的破坏。如果其他一些服务器用这些损坏的结构来升级它们的缓存的话, 这个问题还会加重。

原始的 DNS 查询协议使用了一种非常简单的方法来阻止欺骗。安全 DNS 服务 (DNS service, DNSSEC) 正在不断的部署中。安全问题一直围绕 DNS 在不同类型的标识必须连接时的冲突产生的问题。

13.6 防火墙

密码机制保护传输数据的机密性和完整性。认证协议验证数据的来源。如果要控制允许哪些通信进入系统 (进入过滤), 或者传出系统 (外出过滤), 可以部署防火墙。

防火墙 一种控制网络双方之间的传输流的网络安全设备。

防火墙通常安装在整个组织内部网与因特网之间。它们也常常安装在内部网内来保护单个部门。例如, 一个公司可能会在它的 R&D 部门的子网和公司主网之间安装防火墙。

防火墙应该可以控制网络传输和保护网络。但是所有的传输都必须通过防火墙, 才能进行有效的保护。拨号线和无线局域网是防火墙不能保护的入口点的很好的例子。防火墙可以同意或者阻止访问服务, 在同意访问前验证用户或机器, 以及监视进出网络的传输。防火墙在 20 世纪 90 年代早期开始流行。这时, 很多终端都不能保护自己的 PC 机。它强制将安全中心转移到网络边界上。

防火墙防御一个受保护的系统, 使外部网络中的第三方不能访问那些只能在内部网中可以得到的信息。它们也能限制从里面到外面服务的访问, 这些服务对组织的工作来说是危险的或者不是必需的。防火墙可以决定通过虚拟专用网 (Virtual Private Network, VPN) 路由敏感的通信。VPN 创建一个在组织子网没有直接连接的网关之间的安全连接。所有在两个子网之间的传输都必须通过那些增加了安全措施和密码保护的网关。防火墙也能作为网络地址翻译器 (NAT; RFC 3022), 在公共 IP 地址后面用私有地址隐藏内部机器, 并为内部服务器将公共地址转换为私有地址。

在每个网络层, 我们都能找到用来控制访问的参数。在 OSI 的第 3 层, 我们有源 IP 地址和目的地 IP 地址。在 OSI 的第 4 层, 我们有 TCP 和 UDP 的端口号。许多防火墙假设端口号决定服务, 但这并不是永远正确的。在 OSI 的第 7 层, 有很多与应用程序关联的信息: 邮件地址, 邮件内容, 网页要求, 可执行文件, 病毒和蠕虫, 图片, 用户名和密码, 等等, 而这只是其中的一小部分。通常, 在网络边界的机器可以用来控制对网络的访问, 对离开网络的数据流加密码保护, 或者隐藏网络的结构。

13.6.1 包过滤

包过滤在 OSI 的第 3 和第 4 层工作。这个规则具体地指明哪些包允许通过防火墙, 哪些应该丢掉, 该规则会应用到每个包。典型的规则详细说明了源 IP 地址和目标 IP 地址, 以及 TCP 和 UDP 的源/目标端口号。双向通信规则也可以定义。这样的防火墙可以用 TCP/IP 包过滤发送程序来检查每个包 TCP/IP 的报头, 来实现通过或者丢掉包。

仅仅简单的规则可以强制使用,普通的协议也很难处理。例如,当客户端将 ftp 请求发送到 ftp 服务器时,防火墙不能连接来自服务器为这个请求返回的数据包。我们有针对来自端口 20 的所有包的通用规则,或者从特定 IP 地址来的所有端口 20 的包的规则,但是我们不能动态地定义规则。

13.6.2 状态包过滤器

状态(动态)包过滤器能够明白请求并做出回复。例如,它们知道关于 TCP 开放序列的(SYN, SYN-ACK, ACK)-模式。规则通常只针对一个方向上的第一个包。在第一个包出去之后创建状态。通信中后续的包就可以自动处理了。状态防火墙支持大量协议的过滤规则,而不仅仅是简单的包过滤,例如,ftp, IRC, 或者 H323。

包过滤可以由路由器来做,而且性价比很高。此外,它是非常容易配置提供有限功能的安全平台,而仅仅只有函数性的限制。Linux 系统使用 iptable 的数据结构来定义包过滤规则。可以实施的过滤策略受到参数的限制,参数可以在 TCP 和 IP 头中观察到。过滤规则也可以更容易的规定。

13.6.3 电路级代理

电路级代理有同包过滤差不多的规则,但是并不路由所有的包。规则决定了允许哪些连接,封锁哪些连接。允许的连接会在防火墙和目标之间创建一个新连接。这种防火墙在这里提及到,是由于它很少在实际中使用了。它的函数性和状态包过滤器差不多,但是效能更低。

13.6.4 应用层代理

防火墙可以监控每个应用层协议,代理执行防火墙上协议的服务和客户部分。当一个客户端连接到防火墙时,防火墙的代理就像服务器一样使请求生效。代理是另一个受控调用的例子。例如,一个邮件代理可以过滤掉病毒、蠕虫和垃圾。如果允许客户端的请求,代理就作为客户端,连接到目标服务器。通过防火墙的回应也由代理处理和检查。代理服务器仅仅是外部看见的结构。它在传输中使用排除过滤的方法,例如,去掉邮件的附件。

应用层的代理通常在固化 PC 上运行。应用程序代理可以在进或者出的通信量中提供关闭控制。在这个方面,应用层代理提供了高的安全级别,它所提供的配置比较合适。在底侧,有大量对每个连接的处理,配置也更加困难。在这方面,应用层代理没有这么安全,防火墙安全方面的漏洞也是早就有报道的了。相对于包过滤器来说,它的性能更差,而花费更高。此外,你需要为每个你想保护的服务准备一个代理服务器。因此,面对因特网服务的增长,这条路线并不会太容易。

包过滤器的行为可以比作通过电话号码关闭电话。它们对特定的号码锁定呼叫。例如,收费号码。而应用层代理就像通过监听谈话的电话监视器。

13.6.5 防火墙策略

宽松的策略允许所有的通信,但是封锁一些比较危险的服务(像 telnet 或 snmp),或者封锁通常用来攻击的端口号。这非常容易犯错。如果你忘记封锁一些你该封锁的东西,也许在你没意识到的时候就被利用了。限制性的策略封锁所有的通信,而仅仅开放需要使用的东西,如 http, pop3, smtp 或 SSH。这是一种更安全的做法。如果你封锁了一些需要的东西,就有人会抱怨,然后你就可以允许这些协议。策略通常被看作有肯定和否定表项的访问控制列表。典型的防火

墙规则集应该看起来像这样：

- 允许从内部网络到因特网：http, ftp, SSH, DNS
- 允许从任何地方到邮件服务器：smtp
- 允许从邮件服务器到因特网：smtp, DNS
- 允许从内部到邮件服务器：smtp, pop3
- 允许回应包
- 封锁其他的所有东西

当在实际中部署防火墙时，定义和管理规则集是一个非常重要的问题。

13.6.6 边界网络

邮件服务器应该放置在哪里取决于防火墙吗？邮件服务器需要进行外部访问来从外部接收邮件。因此它应该位于防火墙内部。仅仅是防火墙能保护邮件服务器从外访问的权限。邮件服务器也需要内部权限来从因特网收邮件。因此它应该在防火墙内部。你也许想阻止蠕虫和病毒在你网络上传播，或阻止机密文档泄露你的网络。作为一种解决方案，你可以创建一个边界网络，也被称作非警戒区(Demilitarized Zone, DMZ)。可以用在有服务需要从防火墙的内部和外部都得访问的情况(图 13-12)。除了邮件服务器以外，网页服务器和名称服务器也需要这样做。

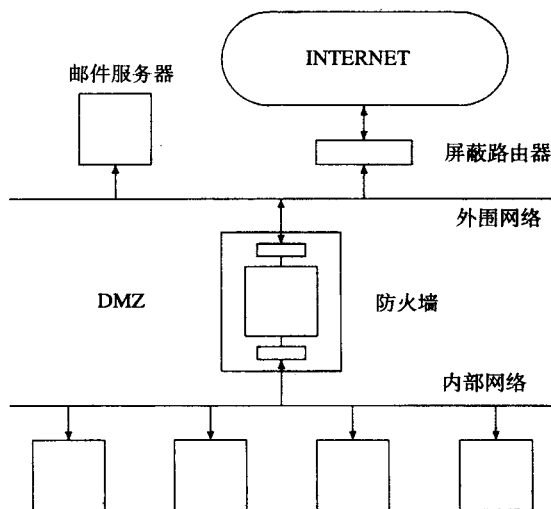


图 13-12 非警戒区

13.6.7 局限性和问题

防火墙不能保护内部不受威胁。封锁服务可能会造成用户的不方便。网络诊断将比较困难。很难支持某些协议。包过滤防火墙不能提供任何基于内容的过滤。如果邮件是被允许的，那么包含病毒的邮件也会被允许通过。甚至应用层代理防火墙也不能完全地检查内容。防火墙不知道操作系统和应用程序的漏洞。今天大量的服务使用 http 的 80 端口。因此，判断通过这个端口的通信是否是合法变得非常困难。

协议隧道，即通过另一种协议发送这种协议的数据，将使防火墙失效。由于越来越多谨慎的管理员除了不得不开的 80 端口外，关闭其他几乎所有的端口，越来越多的协议也通过 http 隧道来穿过防火墙。另一个可选用的隧道是 SSH 协议。

不能检查和过滤加密通信, 因此很多协议像 https 和 SSH 都提供端到端的密码保护, 这样就不能被防火墙监视了。另一种方法是在防火墙中为这类协议使用代理。但是这样就会丧失端到端的安全性。这些发展导致有人宣布防火墙快要不能作为安全结构的部件了, 并且预言安全服务将会从网络转回到终端主机。个人防火墙正在使网络通信上的访问控制转移到终端系统。

13.7 入侵检测

密码学机制和协议是预防攻击的领域。虽然, 这些可以很好地预防所有的攻击, 但是在现实中, 这些攻击也是不太可能的。新型的攻击方式已经出现了, 像拒绝服务攻击(密码学也许还会让该问题更糟糕)。像防火墙这样的边界网络安全设备主要用来阻止来自外网的攻击, 但是它们有时可能做不到。防火墙可能会出现配置错误, 密码可能被窃取, 新的攻击类型也可能出现。而且, 这些设备不能检测攻击发生的时间。因此, 为了检测网络攻击, 可以配置入侵检测系统(Intrusion Detection System, IDS)。

一个 IDS 由一系列收集信息的传感器组成, 这些传感器可以在主机上也可以在网络上。这种传感器网络由一个中央控制台管理。分析数据、入侵报告、触发反应都由中央控制台管理。入侵检测有两种途径: 误用检测(misuse detection)和异常检测(anomaly detection)。传感器和控制台的通信, 特征数据库和产生的日志都应该被保护。而且 IDS 厂家应该提供更新特征库的安全方案。否则, IDS 自身都有可能受到攻击和操纵。已经有很多利用 IDS 本身的漏洞进行攻击的案例。

13.7.1 漏洞评估

漏洞评估用于评价一个网络的安全状态。收集开放的端口, 运行的软件包(哪个版本, 是否打了补丁), 网络拓扑结构等信息, 就可以得到一个不同安全级别的漏洞列表。漏洞评估正如它所基于的技术一样, 必须时常更新才能处理新的威胁。几个安全组织跟踪安全漏洞, 并列出可用的补丁。计算机紧急事故响应小组(Computer Emergency Response Team, CERT)就提供了很好的相关资源, 例如 CMU(<http://www.cert.org/>), SANS(<http://www.sans.org/>)和保存了漏洞档案的“安全焦点”(<http://www.securityfocus.com/>), 还有主要的软件和硬件制造厂家的站点。

13.7.2 误用检测

误用检测(misuse detection)用于寻找攻击特征。攻击特征可以是网络通信中的样式, 也可以是日志文件中用于表明可疑行为的活动。这些特征包括在一台敏感主机上登录失败的次数, IP 数据包可以表明一次缓冲区溢出攻击的一些位, 或者是表明一次 SYN 洪泛攻击的某种类型的 TCP SYN 包。入侵检测系统也得考虑到安全策略, 以及已知的漏洞和攻击的数据库来进行监测。

再者, 这些系统的安全性跟数据库中的攻击特征库联系在一起。新的漏洞经常被发现和利用。IDS 厂家必须跟上最新的攻击和数据库的更新。用户需要安装更新。数据库已知的漏洞和发掘方法会变得大而无用, 而且减缓了入侵检测系统的运行。在写本书的时候(2005 年), 所有的商业入侵检测系统都是基于误用检测的。这些系统也被成为基于技术的 IDS。

13.7.3 异常检测

统计的异常检测(或者称基于行为的检测)利用统计学技术检测可能存在的入侵。首先, 把正常的行为作为基线。在整个操作过程中, 将统计分析被监视的数据和估量背离基线的行为。如果超出阈值, 就发出警报。这种入侵检测系统不需要知道所监视特定系统的漏洞。基线定义常

态。因此,即使没有更新依赖的技术,也可以检测到新的攻击方式。

另一方面,异常检测只检测异常行为。可疑行为不一定会形成一次入侵。在一台敏感主机上的多次试图登录失败可能是由于一次入侵,也可能是管理员忘记密码。这方面的有趣事例可以从 *Bejtlich(2000)* 处得到。攻击也不一定会产生异常。一位细心的攻击者可以在 IDS 监视下“飞过”,而且不被检测到。特别是当基线被设置为动态和自动调整的时候,这种情况就会发生。一位有耐心的攻击者可以一步一步地躲过那个时间,直到他计划的攻击不再产生任何警报。因此,我们必须关心假阳性(假警报)(系统报告有攻击发生,实际上没有任何事情发生)和假阴性(由于攻击行为在正常行为定义的界限之内,导致攻击无法检测到)。

13.7.4 基于网络的入侵检测系统

基于网络的 IDS(NIDS)会在网络通信中查找攻击特征。一个网络适配器以混杂模式运行,实时分析所有经过网络的通信。攻击识别模块把网络数据包作为数据源。有三种常用的技术识别网络攻击特征:式样、表达式或字节码匹配,通过频率或阈值(例如,检测端口扫描活动),以及与次要事件的相互联系(在商用产品中这些事件不多)。Snort 是一种在开源社区中研发的一种流行 NIDS。

13.7.5 基于主机的入侵检测系统

基于主机的 IDS(HIDS)从主机的日志文件中查找攻击特征。它也可以验证关键系统文件的校验和以及在规则间隔时间的可执行性。一些产品可以用正则表达式推敲攻击特征(例如:密码程序的执行和主机文件的改变)。一些 IDS 监听端口的活动,当特殊的端口被访问时产生警报,提供有限的基于网络的 IDS 能力。现在有一种转向基于主机的入侵检测系统趋势。而最有效的入侵检测方法是结合 NIDS 和 HIDS。

由于 IDS 警报已经接近实时性,所以 IDS 可以被用作一个实时响应工具,但是自动响应也不可避免危险。一个攻击者可能欺骗 IDS 响应,但是该响应却指向无辜的目标(使用假的源 IP 地址)。由于 IDS 主动的误报,用户会被强迫退出账户。对管理员的 E-mail 账户重复发送邮件通知会变成拒绝服务攻击。

13.7.6 蜜罐

蜜罐(honeypot)是用于跟踪攻击者和研究、收集黑客活动证据的一种资源。它们模仿真实的系统,其实不包含真正的产品信息。由定义,每一个被监视到的在蜜罐上的活动都是一次攻击。

蜜罐 蜜罐是一个信息资源体系,其价值在于未经授权或非法使用资源(*Spitzner, 2003*)。

蜜罐可以在不同层次上用于与攻击者的交互当中。低层次交互的蜜罐仿真操作系统基本的服务。在这种蜜罐之上攻击者能做的事很少,蜜罐所记录的攻击性行为就很有限。而且,攻击者可能会很快地识别蜜罐的真相,然后逃走。现在也有很多检测蜜罐的工具。蜜罐模仿得越逼真,更多的行为就会被监视到。高层次交互的蜜罐提供真实的服务,但是提供假的数据。攻击者与蜜罐交互得越多,危险越大,因为攻击者会错误地把蜜罐当作台阶去攻击其他机器。

13.8 深层阅读

这一章只描述关于网络安全相关问题和技术的简单轮廓。*Ford(1994)* 和 *Stalings(2003)* 完整地阐述了网络安全, *Atkins(1997)* 解释了因特网安全。*Cheswick, Bellovin and Rubin(2003)*, 以

及 Zwicky, Cooper and Chapman (1995) 在防火墙方面都有很好的著作。Ptacek and Newsham (1998) 阐述了网络入侵检测中不断上升的技术问题。

关于因特网安全的最好的资料来源当然是因特网了。因特网工程任务组的网址是: <http://www.ietf.org>。关于 IPSEC 的文档也可以在 <http://www.ietf.org/html.charters/ipsec-charter.html> 上获得。IETF 规范, 像网络技术一样, 也是在不断变化。对于因特网安全研究来说, 它更是如此, 所以当你阅读这本书的时候, 如果 IPSEC 或者 SSL/TLS 的某些方面已经改变, 也不要失望。

13.9 练习

练习 13.1 ARP (地址解析协议) 将硬件地址和 IP 地址联系起来。这种联系可能随着时间而改变。每一个网络节点均保留了对应 IP 地址和硬件地址的 ARP 高速缓存。高速缓存条目几分钟就过期。节点发现它要寻找的 IP 地址所对应的物理地址不存在于高速缓存中时, 就发送 ARP 请求, 该请求包含了它自己的 IP 地址和硬件地址。ARP 请求中的 IP 地址对应的节点用它的硬件地址响应, 所有其他的节点则忽略这个 ARP 请求。ARP 欺骗的原理是什么? 哪种防御可以用来对抗 ARP 欺骗?

练习 13.2 试设计一个没有状态定义的 TCP 握手协议, 使该协议能够免受 TCP SYN 洪泛攻击。

练习 13.3 客户端 A (a.example.org) 想要通过服务器端 B 的域名 b.example.com 访问。客户端 A 向本地域名服务器请求 B 的 IP 地址。如果本地域名服务器的缓存中没有该 IP 地址, 那么它就会向更权威的域名服务器请求 B 的 IP 地址。该请求包含一个 16 位的序列号。如果收到一个正确的序列号, 那么本地域名服务器则更新它的缓存。其他的回应都会被丢弃。假设一个攻击者向 A 的域名服务器发送请求 B 的 IP 地址的欺骗请求, 同时伪装 B 的域名服务器回应一个假的 B 的地址。欺骗 A 的域名服务器接收假的 B 的 IP 地址, 这个攻击需要多长时间才能成功? (域名服务器可以同时处理多个查询。)

练习 13.4 对于 IPSEC 和 SSL 来说, 运行此协议的节点假设是安全的。为了使这种假设成为真的, 这些节点需要采取哪些附加的安全措施?

练习 13.5 对于一个安全的 E-mail 系统和一台运行安全 E-mail 系统的机器, 你应分别要求哪些安全性能? 哪个协议层最适合提供这个安全服务? 回答时需区分提供匿名服务和不提供匿名服务。

练习 13.6 在设计过滤数据包的防火墙时, 检测 IP 隧道的数据包。

练习 13.7 因特网是计算机病毒的源泉。防火墙保护内部网络免受外部的攻击。防火墙能保护内部网络不受病毒感染吗? 回答时应考虑不同类型的防火墙。在 TCP/IP 层或者应用层的密码保护如何影响防火墙的反病毒能力?

练习 13.8 一个公司允许它的雇员在家或者出游的时候使用便携式电脑。提出一个安全方案保护便携式电脑和公司的企业内联网。

第14章 软件安全

在最近的计算机蠕虫病毒(computer worm)对网络造成巨大损害的时候,或是在广泛使用的软件产品中发现重大的软肋的时候,计算机安全问题便引起了人们广泛的关注。一些攻击通过迷惑用户点击邮件附件,来执行一段攻击性的代码。另一些攻击则在软件开发者所使用的程序抽象层之下,通过使用缓冲区超限(buffer overrun)来操作控制流的方法,把自己攻击的目标放在内存管理的缺陷上。在这两种情况下,攻击者都可以利用提升的优先权来运行一些代码。查找到的程序错误有时很简单,而且排除这些问题也看似很简单。在某些情况下,这种评价是正确的,但是实现一个复杂的系统却又是一件很有挑战性的工作,并且计算机软件的安全漏洞已经有很长的历史了。

在这章中,我们将分析引发操作软件漏洞的常见原因。针对开发者怎样编写安全代码这些详细的介绍,不在研究的范围内。介绍软件安全的方面的书籍参看本章的末尾。

目标

- 介绍一些引发软件不安全性的基本原因。
- 探讨分裂的抽象的危险性。
- 设计软件系统时,可表述出对可用防卫系统的更好的理解。
- 提供恶意攻击的一种分类。

14.1 引言

对于独立的计算机,或是说真正意义上的个人计算机而言,用户能够控制相互发送信息的软件部件。对于连接到因特网上的计算机,不友好的部件之间也可以提供输入。网络软件对于攻击而言是一个普遍的目标,因为它是用于接受外部的输入,并且涉及了一些对内存缓冲的低级操作。如果软件能够有意识地处理一些畸形的输入,从而保护运行时系统的完整性的话,那么该软件就是安全的。安全软件并不等同于添加了安全特性的软件。

14.1.1 安全性和可靠性

软件的安全性与软件的质量和软件的可靠性相关,但是它们之间又有很多不同之处。可靠性处理的是突发性故障。假定根据某种给定的分布,故障肯定会出现。那么基于该分布规则在可靠方面的提高是可以计算出来的。为了使软件更加可靠,使用的测试用例就不能使用通常的典型用法:有多少漏洞并不重要,重要的是这些漏洞被激发的频率如何。在安全的环境中,攻击者会提取输入中的各种分布。因此,传统的测试方法对于发现安全漏洞并不适合。为了保证软件更加安全,就不能使用典型的测试模式(但是也有典型的攻击模式)。

14.1.2 恶意程序分类

拥有破坏性目的的软件称为恶意软件。恶意软件种类繁多,计算机安全中采用拟人的方式对它们进行分类。计算机病毒(computer virus)是一段能够进行自我复制的代码,它利用有效负荷(payload),附着在另一段代码上。这种负荷可以是轻微无害的,比如显示一条信息,或是播

放一段曲子,也可以是有害的行为,比如:删除和修改文件的信息。计算机病毒通过把自己注入到程序代码中,来感染(infect)程序。而蠕虫(worm)能够自我复制却不能感染程序。大众媒体在报道计算机安全事故的时候,并未严格区分蠕虫和病毒。但一段代码在全球范围内蔓延的时候,你可能会阅读到有关病毒攻击的消息,但是事实上,把它称为蠕虫更为准确。

特洛伊木马(Trojan horse)是一段拥有隐密负作用的一种程序,它们在程序文档中并没有说明,用户也没有企图要执行这段程序。逻辑炸弹(loginc bomb)是当一种在特殊激发条件被满足的时候执行的一段程序。

14.1.3 黑客

最初,黑客是指对计算机系统内部细节熟悉的人,他们对计算机系统的使用超过一般用户对计算机的掌握程度。随着时间的流逝,术语“黑客”有了一种贬意,即指某人非法地闯入到计算机系统中。被称为 white hat 的黑客利用自己掌握的技术来帮助软件开发;被称为 black hat 的黑客则非法闯入到系统中,企图创建一些有损坏性或破坏性的东西。可见这两种黑客之间有很大的不同。据报道,普通网络犯罪事件数量在不断地上升。

许多攻击都采用了一种自动和有效的方式,利用已知的安全漏洞(或是设计特征),而并不需要精巧和高深的技术知识。使用别人提供的工具来发起攻击的攻击者被称为脚本小子(ankle-biter, script kiddy)。

在家里尝试危险的代码可能是一种智力上的挑战,但是的确也充满了危险。反病毒专家们已经认识到了把实际操作系统从试验系统脱离开来的重要性。否则,就可能出现代码从实验室泄露出去的危险。你可能会因泄露对其他人的机器执行动作的代码,而触犯相关的法律。因此,通常给你的警告是:不要在家里面尝试这种做法。

14.1.4 环境的改动

改动是安全最大的敌人之一。你可能拥有了一个能够提供非常恰当的安全体系的系统。你改变了系统的一部分。你可能已经意识到了这样的改动会带来安全隐患,但是仍旧这样做了。更糟糕的是,你或许认为改动与安全之间无丝毫联系,这仅仅是一个意外。在看到包含新特性的操作系统版本和对这些新特性中的问题进一步进行修补的版本之间的有规律的交换的情况下,对奇数和偶数版本的操作系统各自的安全性能,就有许多种说法。

14.2 字符和数字

许多软件的缺陷都归结于分裂的抽象。像这种与安全相联系的问题都会在诸如字符和整数这样基本的概论中发现。对于这些问题的描述指的是字符和整数实际在内存中表示的方式。十六制值一般都会加一个前缀 0X 或是前缀 % (一般在 URL 中),反应的是当前使用的不同约定。

14.2.1 字符(UTF-8 编码)

软件的开发者编写的应用程序,应该能够为用户提供到达特定子目录,如 A/B/C 的唯一的访问路径。用户键入文件名作为输入。到达文件的路径名就可以通过 A/B/C/input 这样的应用来构建了。这种限制用户的尝试可以轻易地被绕过。敌人可以通过在目录树中输入 ../ 逐步进入并且通过输入

```
../..../..../etc/passwd
```

来访问口令文件。作为一种应对措施,开发者可执行某些输入校验(input validation),并且过滤

掉这些敏感字符集“.../”，但是这样并不代表就可以一劳永逸了。

Unicode 字符集(RFC 2259)的 UTF-8 编码是被定义为在为 ASCII 设计的系统上使用的 Unicode。ASCII 字符(U0000-U007F)通过 ASCII 字节(0X00~0X7F)来表示。所有的非 ASCII 字符都可以通过非 ASCII 字节(0X80~0XF7)来表示。编码规则的定义如下所示：

```
U000000 - U00007F : 0xxxxxxx
U000080 - U0007FF : 110xxxxx 10xxxxxx
U000800 - U00FFFF : 1110xxxx 10xxxxxx 10xxxxxx
U010000 - U10FFFF : 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
```

其中 xxx 位是 Unicode 数字的二进制表示中最不重要的位。比如：版权号 U00A9 = 1010 1001 就可以用 UTF-8 编码编制成：11000010 10101001 = 0xC2 0xA9。只有最短的可能的 UTF-8 序列才对任意的 Unicode 字符有效，但是 UTF-8 编码器也会接受更长的变量。但多字节的 UTF-8 格式被接受的时候，一个字符就可能得到多种的表示。以下列举的是“/”写法的三种方式。

	格式	二进制	十六进制
1 字节	0xxx xxxx	0010 1111	2F
2 字节	110x xxxx	1100 0000	C0
	10xx xxxx	1010 1111	AF
3 字节	1110 xxxx	1110 0000	E0
	10xx xxxx	1000 0000	80
	10xx xxxx	1010 1111	AF

曾经，在微软的 IIS 中有一个漏洞会引发以下事件：可接受单字节的非法的 Unicode 表示，但是在输入校验中却不检查。一个以

```
{IP 地址}/脚本/..% c0% af../winnt /system32/
```

开始的 URL 就会被直接转换成这样的目录：C：\winnt\system32。因为 %c0%af 是 / 的二字节 UTF-8 编码。因此，..%c0%af.. 就变成了../..。对于这个事，还有进一步的扭曲。现在来考虑一下这个 URL

```
{IP 地址}/脚本/..% 25% 32% 66../winnt/system32/
```

要想知道这个 URL 地址在处理时发生了什么，应先用二进制写出这个序列：%25%32%66。从而你会得到 001000101 00110010 01100110，即 ASCII 字符串 %2f。那么这个 URL 地址就会编码成 IP 地址：/脚本/..%2f../winnt/system32/。现在还没有问题，但是如果这个 URL 被第二次编码的时候，%2f 就会被读成/并且 URL 被直接转换成目录 C：\winnt\system32。在某种程度上，观察的方式改变了字符的意思。

14.2.2 整数溢出

在数学中，整数表示的是一个无限的集合。而在计算机系统中，整数是通过固定长度(精度)的二进制串来表示的。因此只能表示有限个整数。程序设计语言中，整数分为带符号整数和无符号整数，短整数和长整数(或超长整数)。如果计算的结果超过了可以表示的数字，那么就会发生溢出(overflow)现象。在这种情况下，通常我们所熟知的整数计算规则就不适用了。比如，在 8 位无符号整数计算中，需要执行模 256 的运算，所以有 $255 + 1 = 0$ ， $16 \times 17 = 16$ ，并且 $0 - 1 = 255$ 。

带符号的整数是通过二进制的补码来表示的。最高位(最左边的一位)就表明了这个整数的符号。如果符号位是 0，那么这个整数就是正的，它就用正常的二进制表示。如果符号位是 1，

那么这个整数就是负的，要计算 $-n$ 的二进制补码可以执行以下方案：

- 把所有的 1 改成 0，所有的 0 改成 1，来颠倒二进制 n 的表示。对于 8 位二进制整数而言，这一步可以通过 $255 - n$ 来实现。
- 然后在这个中间结果上面加上 1。对于 8 位的二进制整数，这一步可以通过 $255 - n + 1 = 256 - n$ 来实现。数字 256 对应的是进位位。

因此，把一个负的二进制补码加到与其具有相同的模的正整数上，那么两者得到的结果的确是 0。但和一个带符号的整数计算则可能得到意想不到的结果。比如，对于 8 位二进制整数而言，可能会得到 $127 + 1 = -128$ 和 $-128 / -1 = -1$ 这样的结果。在带符号数和无符号数之间转换的时候，一个大的正数就可能变成一个负值： $0xFF = 2^8 - 1$ (无符号) = -1 (带符号)。

在整数表述之间的转换可能会引发安全问题。下面的一条命令就是比较一个带符号的整数变量 `size` 和 `sizeof(buf)` 的运算结果，`sizeof(buf)` 返回的是一个数据类型 `size_t` 的所用字符的大小，比如说可能是一个无符号整数，

```
if(size < sizeof(buf))
```

如果 `size` 是一个负数，并且编译器把 `sizeof(size)` 的结果转换成一个带符号的整数，那么缓存区就可能出现溢出现象。整数的截断可能是另一个潜在的问题。某个 UNIX 版本就有如下的弱点：一个接受 UID 作为输入(假设是一个符号整数)的程序，通过检测 `UID ≠ 0` 来防止对根目录的访问；但是 UID 后来被截取成一个无符号的短整数，使得输入 `0x10000` 变成了 `0x0000` (根!)

在数学中，整数运算的公理就隐含了结果，比如说：若 $b \geq 0$ ，则 $a + b \geq a$ 。在计算机系统中，这种显而易见的例子，就不再是真实的了。这种在抽象模型和实际的实现上的差异就可能引起缓存区的溢出(参见 14.4.1 节)。思考下面的一小段程序代码，这段代码是把两个字符串复制到一个缓冲区中，并且检查两个字符串结合起来之后的长度是否和缓冲区相匹配。

```
char buf[128];
combine(char * s1, size_t len1, char * s2, size_t len2)
{
    if(len1 + len2 + 1 <= sizeof(buf)) {
        strncpy(buf, s1, len1);
        strncpy(buf, s2, len2);
    }
}
```

在 32 位系统中，攻击者能够构造 `s1` 使得 `len1 < sizeof(buf)` 并且设置 `len2 = 0xFFFFFFFF`。接着 `strncat` 就会被执行，并且缓冲区也会溢出，原因如下：

```
len1 + 0xFFFFFFFF + 1 = len1 < sizeof(buf).
```

计算机整数并不是实现数学抽象的整数，而是以 2^w 为模的整数，其中 w 是选作它们代表的二进制 bit 数。这种在抽象层上的断裂(即无关联)可能要导致程序员犯下刚才讨论过的一些错误。

许多程序员似乎都会认为整数有任意的精度，而不会认为整数是使用模数计算的固定规模的数量范围(Ashcraft and Engler, 2002)。

经验教训

如果真的不需要负数的话，那么可以声明所有的整数均为无符号整数。如果计算的是在内存中对象的规模大小，那么你就不需要负数。如果编译器标记了带符号数和无符号数的匹配对，那么需要检查一下是否需要两种不同的表示。如果真的需要，那么请留意要执行的检查。

14.2.3 数组

计算数组使用的是整数计算。如果没有保证这些运算的结果不超过数组长度的检查的话，

那么超过数组的内存区域就有可能被访问。(这是典型的初学者在入门程序课程中所犯的错误。)如果算术运算结果出现负的结果,或是程序员不考虑模加的结果时,都可能会访问一些更低的地址区域。因此当计算数组索引时,就必须得检查上下界。

在下面的例子中,我们假设有一个4字节元素和32位地址组成的数组,那么数组中元素的内存地址就可以通过以下方式计算出来。

元素地址[索引] = 数组基地址 + 索引号 × 元素的大小

如果数组的基地址是0x01720314,并且要在0x0010FF08的内存地址上写数据。我们首先就要计算目的地址和基地址之间的距离,

$$0x10010FF08$$
$$\underline{0x001720314}$$
$$0x0FE9EFBF4$$

并且要让结果除以4来得到索引号0x3FA7BEFD,用十进制表示为1067958013。如果不检查数组索引边界的话,我们就可能在数组的索引位置1067958013处写入我们目标地址的值。

14.3 规范表示

名字(身份、标识)都是广泛使用的抽象表示。我们会给文件和目录取名,以使得可以方便地找到它们。同样地,网络节点在协议栈(DNS域名,IP地址,……)的不同层次也有自己的标识。在安全决策中也需要名字。防火墙可以让通信仅仅在某些节点间来回。安全策略决定用户可以访问哪些文件。如果某一个实体有不只一个名字,或是多个名字代表同样一个事物,那么攻击者就可以通过使用一个等价的名称来绕过安全控制,而这个名称在设置安全策略的时候,是没有被考虑到的。类似这种性质的问题经常遇到。文件名、URL或是IP地址都有不止一种编写方式。

- 文件名: $c:\backslash x\data = c:\backslash y\backslash z\backslash \dots \backslash x\data = c:\backslash y\backslash z\backslash \%2e\%2e\%2e\backslash x\data$
- 打点IP: $a.b.c.d = a \cdot 2^{24} + b \cdot 2^{16} + c \cdot 2^8 + d$
- 符号连接: 指向了另一个文件的文件名。

这些问题对于安全的实施所带来的影响已经口述过了。法律规定使用文件共享服务Napster来封锁对某些歌曲的访问。Napster实现了一个过滤规则,从而封锁了基于歌曲名称的下载服务。Napster的用户可通过使用不同的歌曲名字来避开这项控制。这是一个相当棘手的问题,因为用户决定了哪些名字是相互等价的。

下面我们来考虑不区分大小写的文件名。在这种情况下,myfile和Myfile对于同一个文件来说等价的。把一个不区分文件名大小写的文件系统与一个区分文件名大小写的安全机制体系结合起来。(像这样的事情就曾经发生在Apache的网站服务器和HFS+文件系统上。)假设我们仅仅为一种版本的名字定义了许可,如:Myfile。攻击要求访问myfile,若没有设置任何限制,安全体系就会授予请求,并且文件系统会访问这些资源,而原本这些资源是应该受到保护的。

为了突出强调这些问题,那么在做出文件访问控制决策之前就必须执行规范化(canonicalization)。规范化是解决把同一个名字不同形式转变到单一标准名字的过程。单一的标准名字称为规范名字(canonical name)。任何时候一个对象拥有不同的但却等价的表示时候,规范化都是相关的。

经验教训

不要把接收到的名字当作用户输入,把它们正确转化成标准表示。见“不要相信你的输入”Howard and LeBlanc(2002)。使用规则的方式来定义和确认名字和输入值的有效性。使用完整的路径名,而不是系统自动为你产生的完整路径名。最好不要在应用层上基于名字作出决定,而是使用操作系统地访问控制。

14.4 内存管理

古代的城市和中世纪的城堡都有多层地环形保护墙来防御攻击者。为了击破这些防御设置，攻击者采在墙的下部打隧道的方法来使这些防御墙坍塌。相似的，软件的开发也会设计出逻辑严密的防御体制。如果攻击者能够打通一条隧道直接通往到内存，那么逻辑严密的防御系统也会在底层被削弱。在内存管理上的缺陷可能会露出破绽，让攻击者渗透到系统中去。

在程序设计语言 C 和 C++ 的设计中，程序员可以执行他们自己的内存管理。现在还有一些类似的情形，但是也该轮到开发者自己来正确地管理内存了。像 Java 或是 C# 这样的语言就可以使开发者从这些任务中脱离出来，并且也移除了可能的错误源。

图 14-1 显示了一个典型的内存结构，使用最高内存地址的运行栈，包含了当前调用栈进程中的栈帧。而栈帧又包含了一些信息，如返回地址、局部变量和功能参数。而栈是向内存的低地址处延伸。堆是从内存的最低地址(用户空间)处开始，它动态地包含了要分配的缓存区。堆是向内存的高地址处延伸。

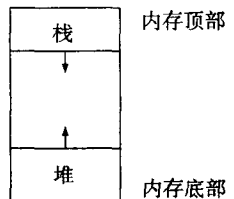


图 14-1 内存结构

14.4.1 缓冲区溢出

用典型的程序设计语言书写的代码中，值是被保存在变量、数组等中的。要执行一个程序，内存区域(缓冲区)就要分配给各个变量。如果赋给一个变量的值超过了分配的缓冲区大小，那么就可能发生缓冲区溢出的现象，并且没有分配给这个变量的区域也会被覆盖。如果这块内存区域已经分配给其他变量，那么这个变量的值就会改变。无意识的缓冲溢出可能使得软件崩溃。如果一个攻击者通过故意赋予一个恶意的值给某个其他的变量那来修改一些安全相关的数据，那么有意识缓冲区溢出就要引起重视了。引起我们注意的是返回地址(说明了下一个要执行的代码片断)和安全设置。缓冲区溢出在曾经一段时间内是安全漏洞的缘由。我们将给出两个历史例子来说明此。

14.4.2 虚拟内存系统(VMS)登录

数字 VMS 操作系统的一个版本在登录程序上有一个漏洞。用户可以通过如下的方式书写他们的用户名登录到他们想要登录的具体设备上。

Username/DEVICE = <machine> (用户名/设备 = <机器>)

在操作系统的一个版本中，并没有检验参数 machine 的长度。如果设备的名字多于 132 个字节那么就会覆盖掉登录启动的进程的特权级。因此用户可以通过提供一个恰当的“机器名”来设置他们自己的优先级。

14.4.3 finger 漏洞

第二个例子是在 1988 年出现的网络蠕虫。在不同的弱点中，蠕虫都会通过在 fingerd 后台程序中开辟一个溢出缓冲区的方式，闯入到运行 UNIX 4BSD 的 VAX 系统中。一条 536 位长的发送给 fingerd 的信息就会被用来覆盖掉系统的栈区：

```
pushl    $68732f      push '/sh, <NUL>'
pushl    $6e69622f    push '/bin'
movl     $p, r10      save address of start of string
pushl    $0           push 0 (arg 3 to execve)
```

```
pushl    $0           push 0 (arg 2 to execve)
pushl    r10          push string addr(arg 1 to execve)
pushl    $3           push argument count
movl     sp, ap       set argument pointer
chmk     $3b          do "execve" kernel call
```

栈已经完成设置，因此在返回到主程序时，就可以执行 `execve("bin/sh", 0, 0)` 命令了，这条命令通过 TCP 打开一条到远程 shell 的连接(Spafford, 1989)。

14.4.4 栈溢出

对调用栈的缓冲区溢出攻击被称为栈溢出。为了表明这种攻击的基本原理，下图 14-2(左图)给出了一个帧的概要。当函数 `function (int a, int b, int c)` 的局部变量 `x` 和 `y` 被调用的时候，这个帧就会被压入到栈里去。开始的是相反顺序的输入值，接着是返回值，保存的帧指针，它指向之前的栈顶。最后，分配的是函数中定义的任何局部变量的缓冲区。(这种精确的栈调用是操作系统所特有的。)

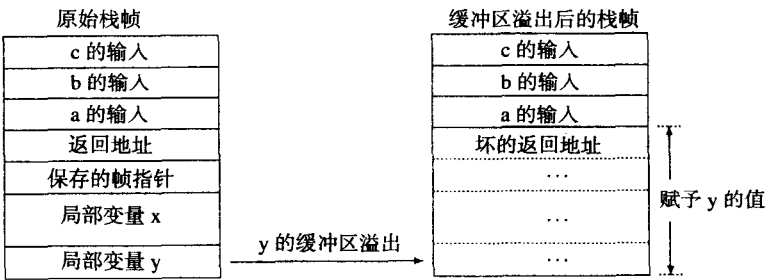


图 14-2 栈的粉碎性攻击

假如在函数中不检查赋给 `y` 的值的规模大小，那么攻击者就可能赋给 `y` 一个值，使它超过分配给 `y` 的缓冲区域，从而覆盖了在 `y` 之上的一段内存区域。特别是，攻击者可以用他的代码的开始地址重写返回地址(图 14-2)。要注意的是，如果栈是从内存的低地址区向上延伸的话，就不会发生这样的攻击现象。

对于攻击者而言，最后的任务就是寻找一种方式，使得攻击代码能够渗透到系统中来。一种简单的方式就是把代码发送到一个输入变量中，这样就可以使得它存储在栈中。当攻击者不能准确地预测到输入变量和攻击代码存放处的时候，那么在代码的起始处的 NOP(空操作)就会弥补代码位置的变化。

14.4.5 不可执行的栈

我们将会在 14.7 节中讨论缓冲区溢出的一般性防御方案。这儿，我们只提及两个特定栈的防御策略。使用不可执行的栈(nonexecutable stack)体系可以防止攻击代码被放入到栈里面去。代码也不会再次被编译，但是要求可以执行栈的现有软件也不能工作了。

可以在返回地址的下方内存区域中设置一个警戒灯的(随机的)预警值[⊖]，这样试图改变返回值的行将会被检查到(Cowan et al, 1998)。在返回之前，系统将会检查预警值是否还是正确的(图 14-3)。为了使用这种技术，必须再次对代码进行编译，这样一来，对预警值的插入和之后的检查便会被加入到对象代码中。

⊖ 在煤矿中用警戒灯来检查天然气的泄漏，如果它灭了，就意味着要矿工需撤出。

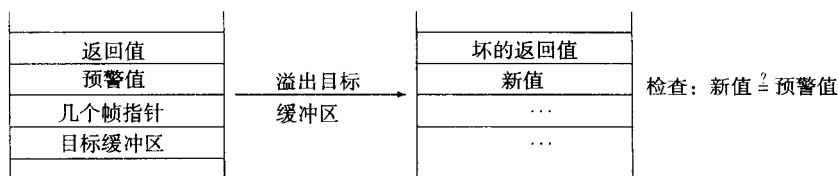


图 14-3 预警值用来暗示栈可能的溢出

14.4.6 堆溢出

由于基于栈溢出攻击的问题过于突出，设计者们对于栈的保护投入了比对堆的保护更多的精力。堆是通过应用程序动态地分配出去的一段内存区域。堆中的缓冲区溢出称为堆溢出。在堆中更难判断怎样重写一段缓冲区域，同样在进程中，哪一段缓冲区域被覆盖了也很难确定。如果你是一个攻击者，那么你很可能不想在你占领系统之前就摧毁它，但是即使一直没有成功的攻击行为，对于防御系统来说也是一种威胁。

为了控制整个执行，攻击就不得不覆盖掉影响控制 and 数据流的一些参数。这些参数在堆中也是可以找到的，比如：指向文件的指针、指向函数的指针。选取一个容易受到攻击的程序，在这个程序中，定义了待写入数据的(临时)文件。然后创建一个缓冲区溢出，它使用一个指向目标文件的指针来覆盖掉了指向暂时文件的指针，其中目标文件可以是口令文件之类的。现在，程序就可以访问目标文件了，而且攻击者还可以对目标文件进行写操作。函数指针 `int (*function)(char *str)` 允许程序员动态修改要调用的函数。执行时，通过指针指向的函数就会被调用。当一个缓冲溢除覆盖掉了函数指针时，那么由攻击者定义的函数就会被执行。函数指针的利用需要一个可执行的堆。

攻击代码可以作为一个参数传递给程序，这样它就可以存储在栈中。这种方法需要一个可执行的栈。或者是，攻击者可以估计一段距离(偏移量)，这段距离就是发生溢出的目标缓冲区地址和堆中 `shellcode(/bin/sh)` 位置之间的距离。这种方法用于可执行堆。堆被执行的可能性要比栈大得多。“堆溢出”原来仅仅做为理论兴趣研究，而现在到处都可以看到它们的身影。

经验教训

重定向指针是攻击系统非常有效的方式。

14.4.7 类型混淆

用类型安全(内存安全)的语言写的程序，不能用不恰当的方式访问内存。一个著名的例子就是 Java 程序语言。每一个 Java 对象都是一个类的实例。我们只可以操作一个给定类的某些对象。Java 虚拟机通过使用带对象类标记的指针在内存中跟踪对象。静态的和动态的类型检查都应避免对类和对象提供抽象型的访问。自动垃圾清理任务机制，会进一步对内存进行管理。

一个类型混淆 (type confusion) 攻击，可以操作指针结构，使得两个指针指向同一个对象，其中一个指针有错误的类型标记。因此这个对象就会被能够访问这个错误类型的第三方操控。比如，假设有一个可信的对象 A，类型是 *Tr*。但一个攻击者成功地用一个不可信的 *Un* 类型对象 X 的指针来指向内存中同一段缓冲区的时候，那么可信对象 A 就会像类型 *Un* 一样被修改掉(图 14-4)。常常(并不总是)，一个类型混淆攻击可能延伸到整个操作系统，并且可以随意地修改掉对象。类型混淆攻击很少发生，但是它们的确发生过。Sun 安全公告 #00218(2002 年 3 月 18 日)

就预警过 Java 虚拟机执行上的一个问题。在 2004 年的时候, 还报告过 Java[⊖] 的移动电话上的一个攻击。

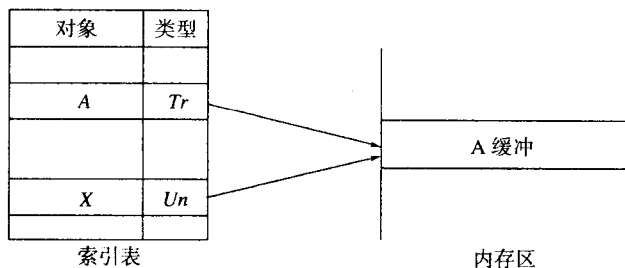


图 14-4 类型混淆冲突

网景公司 3.0 β 5 版本的 Navigator 就有一个漏洞, 这个漏洞允许简单的类型混淆攻击 (McGraw and Felten, 1997)。但是这个漏洞在 3.0 β 6 版本上被补上了。Java 程序中允许使用类型 T, 和数组类型 T。定义数组类型仅仅为内部使用, 它们的名字也是以 [符号开头的。程序员自己定义的类名是不允许以这个字符开头的。这样就不会有冲突的危险性存在。但是一个 Java 的字节码文件可以把他自己的名字申明为一个专门的数组类型名字, 这样就可以重新定义一个 Java 数组类型。

GovinDavajhal and Appel (2003) 描述了一个使用随机内存错误、设计精巧的类型混淆冲突。这种攻击策略用来创建一个数据结构。在这个数据结构中, 一个随机的内存错误很可能改变某个引用, 这样它就可能指向一个错误的对象。为了达到这个目的, 我们现在来创建两个类 A 和类 B:

```
class A{
    A a1;
    A a2;
    B b;
    A a4;
    A a5;
    int i;
    A a7;
};

class B
    A a1;
    A a2;
    A a3;
    A a4;
    A a5;
    A a6;
    A a7;
};
```

攻击的过程如下:

(1) 用类型 B 的许多对象装满内存区, 并且在位置 x 处用类型 A 的一个单独的对象填充。使得 B 中所有的 A 成员都指向这个 A 对象。

(2) 不断地扫描 B 对象, 并且检查它的 A 成员域, 直到检查出一个内存错误出来为止。在内存的芯片上点燃一盏明亮的灯泡, 是一种已证明的引起内存区错误的方法。

(3) 假设在 b 对象的字段 a 处的 i 位上, 出现了转换。那么创建一个 A 类型的指针 r , 并且设置 $r = b'.a'$ 。

(4) 创建一个 B 类型的指针 q , 并且把 r 的 B 字段作为 q 的引用, 比如: $q = r.b$ 。

如果有内存错误, 那么 r 就会包含地址 x , 而且我们也可以通过从地址 $x +$ 偏移量的方法得到 q 的值, 这个偏移量 (offset) 指的是一个 A 对象的基地址和它的 b 成员变量的起始地址之间的一段距离。但是, 因为存在内存错误, 就会使得 r 包含地址 $x \oplus 2^i$, 并且只能从 $(x \oplus 2^i) +$ 偏移量得到 q 的值。给出攻击中的内存布局, 就很有可能得到 A 对象的地址 x 。现在有一个 A 类型

⊖ Adam Gowdiak: Java 2 的一个很微小的安全漏洞, 在 2004 年马来西亚吉隆坡举行的安全会议上有所讨论。

的指针 p ，它包含了 x ，同时有一个 B 类型的指针 q ，它也包含 x 。这样，类型混淆攻击就完成了。

经验教训

对硬件层的攻击可以削弱这个层之上的安全控制。

14.4.8 疯狂的电脑黑客

有时候，一个软件的开发者可以跨过抽象层，而直接控制内存的条目。错误也可能导致漏洞，这一点可以在下面一个很有历史意义的故事中说明。在这里受到影响的操作系统是 ICL 的 VME/B 操作系统。下面我们将逐一讲述该故事。

VEM/B 在文件描述符中存放文件信息。用户：STD 拥有所有的文件描述符。但是如果一旦文件描述符在不同的安全层次上被分类了，那么这就会有麻烦。正是因为这个原因，STD 不能访问分类的文件描述符。结果是，这些文件描述符就不能在正常的备份中被存储下来。解决方式很简单。创建一个新的用户：STD/CLASS，让它常管分类的文件描述符。这种特性因此就很快地在升级的系统中被包含了。

用户：STD/CLASS 除了拥有文件描述符，再也不会拥有其他的目的了。因此，对于任何要想使用 STD/CLASS 登录的人来说，这个名字并不是他们所期望的，也不是必要的。为了使登录不可能，STD/CLASS 的口令可定义成 Return 键。没人能够登录，因为 Return 经常会被解释成为口令的分割符，而不是口令的一部分。用户的配置文件中，口令是通过修补 16 进制代码来设置的。不幸的是，错误的字段被修改了，出现了一个不能识别安全级别的用户，而不是一个不能登录的用户。不能识别安全级别会被解释成为“不安全”，因此设计者们就不会达到他们的目的。

不过仍然留下了一道防线。用户：STD/CLASS 只能从主控制台上登录。但是一旦主控制台掉电了，控制连接的下一个设备就会被认为是主控制台。

这个缺陷是被一个黑客所开发出来的，当时他自己也在操作 VME/B 系统。因此他自己就有足够的机会来进行细节分析和试验。他在夜里当没有人坚守的时候，通过拨号上网的方式闯入许多大学计算机中，修改和删除系统和用户文件，留下一些来自 The Mad Hacker 的消息。但是他成功地跟踪了，并且被带到了法庭里，被判入狱（根据英国 1971 年的犯罪条款）。这是英国第一起黑客判罪，在 1991 年得到了地区法院的支持。

14.4.9 AS/400 机器接口模板

AS/400（应用系统 400）被认为是 IBM 的一款操作系统，它在财政部门的中等计算机中很流行，也被认为是比较安全的。在 AS/400 系统中，开发者很容易访问到操作系统以下的层。被作为是机器接口模板（machine interface template）的机器语言程序，并不受制于操作系统的安全控制之中。机器接口模板起初被有经验的 AS/400 开发者用于提升软件性能。但是，有了这种技术后，也有可能修改掉（或覆盖）对象，从而改变安全设置。

IBM 起初通过移除一些具有破坏性质的机器接口命令，并使用一个语法检查器来检查那些在机器接口模板下的命令，试图停止这些修改。语言检查器指的是一个合法命令表。一旦软件开发者发现了检查是怎样执行后，他们就会简单地用一个包含所有原始命令的表来替换 IBM 的表。下一步就是保护操作系统的完整性，AS/400 把用户态和系统态分开，同时把用户区域和系统区域分开。在用户区域的系统态程序就会提供给用户访问受保护软件的权利，这就像数据库管理系统一样。关于技术上的讨论，以及更多的关于 AS/400 的信息和在操作系统安全上更好的建议可以在 Park(1995) 中找到。

14.5 数据和代码

在评估一个系统的安全性时，数据和代码可能是重要的抽象概念。但一个以输入形式出现的数据被执行时，这个抽象就会被打破，攻击也可能发生。下面的例子中都可以说明，作为输入形式而接受到的数据可能不是程序员所期望的。

14.5.1 远程登录漏洞

UNIX 的登录命令格式是：

```
login[-p][-h<host>][[-f]<user>]
```

其中-f选项强制登录。用户也不必输入口令。而 rlogin(远程登录)命令允许用户登录远程的机器。这条命令的格式是：

```
rlogin[-l<user>]<machine>
```

而远程登录的守护进程占用这个命令中的第一个参数，并且会把登录请求发送给在第二个参数中命名的主机。某些 Linux 和 AIX 版本不会检查名字域的语法。所以请求

```
rlogin -l -froot machine
```

就会导致

```
Login -froot machine
```

即，在指定的机器上，使用一个强制的登录作为根目录。这个问题是由两条命令的复合引起的。每条命令本身并没有漏洞。而远程登录进程的语法检查能避免这个攻击。

14.5.2 脚本

脚本语言是常用于书写网络应用程序的一种解释性语言。这种语言有：Perl、PHP、Python、Tcl 或是 Safe-Tcl。公共网关接口(Common Gateway Interface, CGI)是一种标记性语言，它把统一资源定位器(Uniform resource locator, URL)和 html 形式翻译成可执行的程序。这些程序可以是任何满足一些标准的 CGI 需求的语言书写的，包括先前提到的那些语言。图 14-5 描述了 CGI 的基本规则。客户端发送一个 URL 和 html 表格给服务器，指定一个 CGI 脚本和它的输入参数。这种请求就会被请求就会转化成用户的服务器的程序可以执行的程序。网络服务器可以调用应用程序，如服务器端包含(Server-Side Include, SSI)。使用 SSI，在服务器上的文档就可以包括系统命令，这称为 SSL 协调。当客户端请求一个这样的文档后，这些系统命令就会被评估，结果也会被插入到一个返回给客户的文档中去。



图 14-5 服务器执行 CGI 脚本

一个简单的例子即可以说明 CGI 脚本会怎样引起破坏。一个发送给服务器的脚本可能像这样：

```
cat thefile | mail clientaddress
```

在这里 thefile 是文件的名称，clientaddress 是客户端的邮件地址。一些恶意的用户可能会输

入 user@ address | rm -rf 作为邮件地址。服务器就会执行

```
cat thefile | mail user@ address | rm -rf /
```

而且，在把文件邮给用户后，服务器删除所有脚本允许的删除的文件。就像在任何其他可以控制的请求中一样，对 CGI 脚本的输入也应该进行过滤。被调用的操作越强，你就越应该对要接收的输入小心仔细地查看。SSI 可能会非常的强大。SSL 协调的具体格式是：

```
<!-- # operator arg1 = "string1" arg2 = "string2"..... ->
```

最后的适应性是操作符号 exec 使用参数 cmd 来提供的。SSL 在线

```
<!-- # exec cmd = "myprogram myparameters" ..... ->
```

传递为执行提供了字符串 myprogram myparameters to /bin/sh。(这个选项可以通过在服务器的多个选项中声明 Options IncludesNOEXEC 来禁用。)恶意可以来自于程序中，如果 myparameters 包括一个 shell 转义符的话，也可以是来自于一个不相关的程序的参数。不转义操作通过注解转义字符[○]的方式，去除了来自客户端的 shell 转义字符。命令：

```
unescape "string1; string2"
```

返回的是 "string1...string2"。像 Perl 这样的脚本语言，也会作一些不转义操作。但是对转义字符的过滤必须知道使用集中所用的转义字符，这是一件很麻烦事情，正如 CERT Advisory CA-2002-2 中讲述的一样。

14.5.3 SQL 插入

SQL 是广泛使用的一种数据库查询语言。应用程序可以处理来自网页中的 SQL 查询来访问数据库，比如在交互网站上创建动态网页。恶意的用户输入，会被处理来破坏开发者的设计意图。考虑一下下面的对客户端数据库的查询：

```
string sql = "select * from client where name = ' " + name + " ' "
```

原本的意图是处理类似：select * from client where name = 'Bob' 的查询。但是当攻击者输入 Bob' OR 1 = 1-- 时，查询就会被这样处理：

```
select * from client where name = 'Bob' OR 1 = 1-- '
```

在读入参数 WHERE 字句的时候，要进行一些逻辑上的分割，“name = 'Bob'”和“1 = 1”，还有-- 会被当成注解的起始部分进行读入。参数假定是正确的，所以这样的分割也认为是正确的。整个客户端数据库都会被选取。另一个恶意的输入是：Bob' drop table client --。

作为一种应对措施，我们会过滤掉一些非法的字符串输入，并且用安全的字符来代替它们。在我们的例子中，单引号(用来结束一个参数)可以被双引号(用来结束查询)替代，这样刚才描述到的攻击就会被阻止。但是，这不会让人满意的，因为若要阻止一类攻击，那么就必须知道所有的非法字符。

经验教训

不要试图去了解哪些输入是不正确的。而应该定义哪些输入是正确的，例如，使用规则的表达式，并只接受正确的输入。

14.6 竞争条件

当多个计算操作访问共享数据，而且这些操作的结果会依赖于访问的顺序的时候，就创造了

○ 转义字符意味着它后面的数据会在不同的上下文中被解释。

竞争条件。就像在 Java servlet 中一样,当有多个进程,或是在一个多线程的进程中有多个线程,要访问同一个变量的时候,这是可能发生的。攻击者会在一个变量被核实后而又在受害者使用它之前,试图利用竞争条件来改变这个变量。这个话题在安全文献中被称为是 TOCTOU (time of check to time of use)。

我们给出一个在 20 世纪 60 年代有关 CTSS 系统(一个早期的分时系统)的历史性例子。我们的故事的如此开始 (Corbato, 1991):

曾经,一个用户发现口令文件被看成是“当天消息”。

发生什么了呢?在 CTSS 中,每个用户唯一对应着一个自己的根目录。但一个用户调用了编辑器后,该目录下都会创建一个草稿文件 (scratch file)。这个草稿文件有一个固定的名字,比如说 SCRATCH,当然具体取决于正在编辑的文件的名字。这是一个很合理的设计思路,因为一个用户每一次只能运行一个应用程序。其他任何人都不能在另一个用户个人的目录下工作。因此,到目前为止,还没有必要为编辑器提供一个以上的草稿文件。而且,系统也会被看成一个拥有它自己目录的用户。在同平台上,几个用户会作为系统管理员同时工作。在同一个时间,允许多个系统管理员同时工作(访问系统目录),这看起来十分的方便。现在来演示如何执行这个特点(图 14-6):

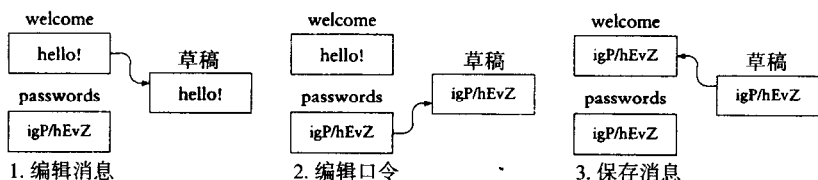


图 14-6 在 CTSS 中共享草稿文件

- (1) 系统的管理员开始编辑一天的信息: SCRATCH: = MESS;
- (2) 然后第二个系统管理员编辑口令文件, SCRATCH: + = PWD;
- (3) 并且第三个系统管理员存储编辑的文件。所以我们将得到 MESS: = SCRATCH = PWD。

经验教训

原子事务 (atomic transaction) 是一种操作的一种抽象,这种操作应该作为一个单独的单元被执行。当这种操作在执行的时候,不允许其他任何操作发生。这种抽象是使用锁机制来实现的,它能够包含事务中所需要的访问的资源。锁机制的一个简单的例子就是 Java 中的同步 (synchronized) 关键字。在 Java 中,保护原子事务很重要,因为 Java 是多线程程序。但是这项任务会留给程序员来完成。添加同步机制的弊端是:会影响性能。

14.7 防御

我们可能会单独地对待出现的问题,并且寻找专门的解决办法,这仅仅限于特定的程序设计语言和运行时系统上。这实际上是在元级层次上的渗透和补丁 (penetrate-and-patch)。相反地,一般我们应该能辨别出一种通用的模型来。在安全软件的情况下,那些重复的模型是一些熟悉的程序抽象,如变量,数组,整数,数据和代码,地址(资源定位器)或是以一种可能破坏抽象的方式执行的原子事务。

软件安全问题存在于处理器结构中,我们使用的编程语言中,我们遵循的代码规范中,编译时检查中(如 canaries),以及软件开发和配置中。

14.7.1 防止：硬件

很多缓冲区溢出攻击会覆盖控制信息。这样的攻击通过硬件特征来保护控制信息就可以防止。Intel 的 Itanium 处理器就有一个专门用于返回地址的独立的寄存器。R. B. Lee et al. (2003) 描述了有独立的安全返回地址栈 (SRAS) 的处理器结构。在硬件级加入保护机制时，没有必要重写或者重新编译程序。只需要修改一些处理器指令。但是，这样做除了需要新的硬件以外也还有一些缺点。现有的软件可能不能正常工作，如：栈中的可执行代码或者多线程代码。

14.7.2 防止：类型安全

我们可以通过使用一种防止我们犯错误的程序设计语言来阻止软件漏洞。类型安全 (type safety) 可以保证我们不会出现一些捕获不到的错误 (Cardelli, 1997)。类型安全语言包括 Java 和 C#。语言不必保证类型安全，比如 LISP 语言。安全可以通过动态检测和动态运行时检测来保证。编译时静态类型检测是指检测操作数在执行期间得到的参数是否都是正确的。动态类型检测要比运行时静态类型检测复杂得多。但是它会导致更快的执行效率，因为困难的工作提前就已经完成了。

对与开发者而言，类型安全常常意味的仅仅是“内存完整”。而对于研究理论的人来说，类型安全意味的是“执行的完整性”，并且后者现在更加相关些。有了类型安全后，我们就能够靠近“近乎安全的软件”，但是也存在一些警示。很难完整地证明类型安全，而且也有可能会有一些问题隐藏在实际执行的时候 (参见 14.4.7 节)。更加困难的问题是，必须精确地定义执行完整性对与一个给定的软件成分 (如操作系统) 来说意味着什么。因此，我们经常使用的类型安全性对于安全而言很有用，但是这并不表示一定就是安全的。

14.7.3 预防：更安全的函数

C 和 C++ 中，作为一个软件开发者，你必须避免写一些可能会导致缓冲超限的代码。C 就是因为它的字符串处理函数，例如 strcpy, sprintf 或 gets 而臭名昭著的。下面以 strcmp 为例来说明：

```
char * strncpy( char * strDest, const char * strSource );
```

如果源缓冲区或是目标缓存区是空的话，那么就会出现一个例外。如果字符串不是空结束的话，结果就不确定了，而且也不会检测目标缓冲区是否足够的大。我们用更安全的函数来代替不安全的函数是明智的，在安全的函数中，要处理的字节数或字符数都应说明清楚，比如 strncpy, _snprintf 或 fgets。函数 strncpy 的说明如下：

```
char * strncpy( char * strDest, const char * strSource, size_t count );
```

使用安全的字符串处理函数，它本身并不能根除缓冲超限。你还必须知道字节数，并且保证计算是正确的。你必须知道数据结构的精确的最大长度。当各个数据结构都在一个函数中使用的时候，这可能会比较简单的，但是如果各个数据结构在多个程序之间共享的时候，那么这就可能会比较困难了。如果你低估了缓冲区所需的长度，那么代码可能会不可靠，甚至崩溃。可以配置一些编译器来检测那些不安全的函数。

14.7.4 检测：代码检查

人工代码检查非常繁琐，而且也容易出错，因此希望让它实现自动检查。代码检查工具会扫描代码，寻找那些潜在的安全问题。Ashcraft, Engler and Hallem et al. (2002) 就介绍了一种对 C 源代码使用元汇编的工具。它可以作为一个专家系统，把多个知名的安全期刊上规则集成起来。

这些规则会寻找一种这样的模式：

“不可信源→洁净检查→可信汇点”

如果一个不可信输入在没有经过恰当的检查而直接进入汇点的话，那么就会出现警报。这种模式也可以用来学习新的规则。比如说，如果代码分析只是一个检查，而没有汇点的话，那么很可能这种规则不知道汇点的具体种类。如果的确是这样，并且一个新的汇点已被辨别出来，那么相应的规则就必须添加到规则库中。代码检查可以很轻易地捕获到熟知的那些问题，但是也不能保证就没有它自己的漏洞，或者是那些被标记成问题的就一定是错误。在实践开发代码分析工具的时候，要保证尽量地降低发生错误报警的几率。

14.7.5 检测：测试

测试也是软件开发中的一个完整的部分，一般用来证明函数的正确性。在安全测试中，情形稍有不同，因为我们必须指出安全函数的正确性。这就意味着我们必须知道潜在的威胁，分析威胁后，还要提出一些安全测试所需要的输入数据。测试也不能保证没有错误出现，同样也不能作为证据。证据仅仅表明我们在所用的抽象中，寻找到那些错误似乎没有的。

在白金测试(white-box testing)中，测试者知道源代码的各个组成部分，并且能够使用这些信息来进行测试。黑盒测试中，测试者并不知道源代码。黑盒测试(black-box testing)的一个著名的例子是对简单网络管理协议(SNMP)执行的安全分析，这是由 Oulu 大学安全程序组完成的(参见 CERT Advisory CA-2002-03)。

经验教训

你并不需要源代码来观测内存是怎样使用的，或测试输入能否恰当的检测。实际上，知道一个软件成分高层部分的说明和它的接口，要比只拥有源代码有用的多。

对于安全测试最重要的也是通用的一项技术就是数据突变(data mutation)。(Howard and LeBlance, 2003)。数据突变是向程序接口发送的异常的输入数据。对于安全测试来说，随机的输入并不是特别有用。他们测试究竟那些不可预知的数据是怎样处理的，但是通常情况下，并没有很多代码会被测试，因为许多输入会被简单的输入检查过滤掉，或者这些输入可能和代码相互冲突。(当代码发生冲突的时候，就要检查是否返回的地址已经被输入数据覆盖掉了，这表明代码也容易引起缓冲超限。)也有部分的输入会试图穿过第一层防线，然后去检验更多的代码。

对于数据容器来说，数据突变可能会引起以下一些情况出现。程序创建的容器已经存在了。如果软件试图创建一个新的临时文件，但该文件又已经存在的话，那么软件就可能会崩溃。而程序试图访问的容器不实际存在。比如说，在 Windows 中的 NULL DACL。如果没有 DACL 的话，访问允许就不会被检查，并且访问会被授予权限。程序也就不能访问容器了，或是访问就会受限制。设置一些允许权限，这样就可以使得测试的组件不能访问容器，或是提供不充分的访问权限。如果容器的名字改变了，测试一下会发生什么，并且使用那些与其他文件相联系的文件名来测试一下代码，看看又会发生什么。

对于数据而言，测试案例应该包括空(就是无值，看看如果输入字段为空，会发生什么)，零，错误的符号(它致使 Ariane 5 火箭的坠落^①)，错误的类型，超出界限的输入，以及有效和无效数据混合的输入(利用有效的数据来输入检查，这样就又可能处理无效的数据)。测试案例也应该包括那些输入过长，或者输入过短的情况。如果一个应用程序要求固定长度的输入，而输入

① 见 <http://archive.eiffel.com/doc/manuals/technology/contract/ariane/page.html>。

的数据长度得不到满足得情况下,安全问题就又可能发生了。比如, *Graff and Wyk(2003)*, 就描述过 Sun tarball 的漏洞。对于网络代码,测试案例还应该包括之前协议运行的回复消息,或是非同步消息,零碎的消息,部分正确的协议机制(用来测试入 TCP SYS 之类的泛洪攻击),和高通信量。

14.7.6 缓和:最低权限

在两种情况下,可以使用最低权限规则。当写代码的时候,少用一些必须的权限来运行代码。如果可以使用尽可能少的权限来运行代码的话,那么破坏也就会更少。当在拓展一个软件系统的时候,也会用到最低权限。不要给用户必须权限以外的权限。不要激活那些你不需要的选择。

在 UNIX 中,发邮件程序(sendmail program)可以说明后面的那个问题。配置邮件系统的系统管理者需要确定所有的消息都能顺利地到达它们目的地。如果在网络节点处的邮件配置能够被检测到,并且可以通过远程修改,而不需要系统管理员通过登录节点,那么这样做就会很有用。为了达到这个目标,在发送邮件程序中包括了一个调试(debug)选项。当这个选项转向它的目的地后,在邮件消息中的用户名就会被发送邮件程序在目标系统上执行的代码所代替。这种特征是因特网上蠕虫攻击的一方面。

经验教训

过去,软件一般都是在在一个开发的体系下使用的,附带有各种的访问允许权限和各种可以激活的特征。应该轮到用户通过移除特征和附加一些受限制的访问权限来使整个系统得到巩固了。今天,软件更像是在一个封闭的系统下使用的。用户需要开启他们需要的特征。

14.7.7 反应:紧跟时代步伐

当发现一个软件的漏洞时,就必须先确定受影响的代码,同时需要测试一个修正版本,用户也需要将新的补丁程序安装到原来的软件上,因此不仅仅是软件开发商需要对他们所发现的问题作出及时的反应。同时用户也需要采取行动,首先要知道问题在哪里。根据 *Arbaugh, Fithen and McHugh(2000)* 的调查显示,绝大数软件的漏洞被暴露的时候,相应的应对措施已经找到了。图14-7是来自于这样的报告。补丁对于潜在的攻击来说,也算是一种资源,因为他们提供了已经被移出的漏洞的一些信息,然而在其他有些地方,这些漏洞可能仍然还存在。因此很有必要讨论一个何时揭露漏洞和发布补丁的恰当方案。

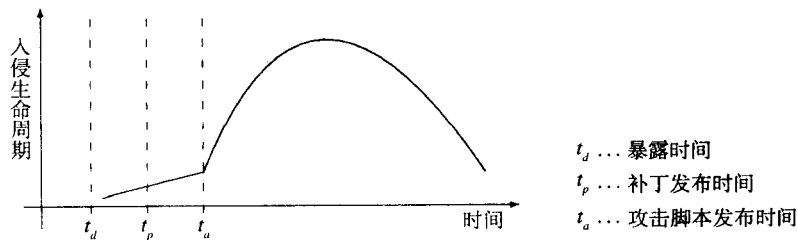


图 14-7 入侵的生命周期(依据 *Arbaugh, Fithen and McHugh(2000)* 的调查。© 2000 IEEE)

14.8 深层阅读

关于编写软件安全的书,现在已有很多的选择,比如: *Viega and McGraw(2003)*, *Harward and LeBlanc(2002)*, *Graff and van Wyk(2003)*, *Viega and Messier(2003)* 以及 *McGraw(2003)* 的

著作。各种缓冲溢出攻击的技术细节 *Foster(2002)* 也描述过了。如果你对攻击背后的详情有兴趣的话,那么你可以看看 *Stoll(1989)*, *Shimomura(1995)* 和 *Dreyfuss(1997)* 的著作。对于一般的读者而言,可以查看里面很多的实际的事例。你可以通过 CERT 顾问获得很多新的攻击信息,可以发邮件给某些网站(如 <http://www.securityfocus.com>)或是到软件商的公告栏里面得到最新的信息。*Phrack* 杂志列出了软件漏洞的许多技术细节。对这个领域的历史感兴趣的人,可以参看一下第一篇关于计算机蠕虫的论文(*Shoch and Hupp, 1982*)。

14.9 练习

练习 14.1 下面这个循环会终止吗?如果可以的话,那么是在多少次循环后呢?如果变量的类型是无符号整型而不是带符号的整型,结果会改变吗?

```
int i=1;
int c=1;
while(i>0)
{
    i=i*2;
    c++;
}
```

练习 14.2 对于在 14.2.2 节中讨论的连接两个字符串的缺陷,给出一个改进的措施。

练习 14.3 对于一个预警机制的另一种可以代替的办法是,把 NULL 值放在返回地址的下面的。这种设计的基本理由是什么?这种办法在什么时候可以达到它的目标,它的不足之处又是什么呢?

练习 14.4 下面这段代码采用零填写缓冲区,它有什么缺陷?怎样解决这个问题?

```
char * buf;
buf = malloc(BUFSIZ);
memset(buf, 0, BUFSIZ);
```

练习 14.5 UNIX 系统中,可以使用环境变量来设置应用程序的行为。一个调用另一个程序的程序,可以为被调程序设置环境变量。在攻击中,怎样利用这个事实呢?程序开发者又该应用什么措施来防御呢?

练习 14.6 假定有一个内存系统,它用一个双链表 IDX 以升序的方式存储空白块。并且使用下面的宏 frontlink 来将大小为 S 的空白块 P 插入到链表中去。可以通过 unlink 函数把空白块从列表中取出来。如果有一个块,它已经在空白块列表中,却要求被加入进去,而且之后还要从列表中移除,那么系统会有什么样的反应呢?

```
#define frontlink(A, P, S, IDX, BK, FD)
.....
[1] FD = start_of_bin(IDX);
[2] while (FD! = BK && S < chunksize(FD))
{
[3]     FD = FD->fd;
}
[4] BK = FD->bk;
[5] P->bk = BK;
[6] P->fd = FD;
[7] FD->bk = BK->fd = P;
}

#define unlink(P, BK, FD) {
[1] FD = P->fd;
[2] BK = P->bk;
```

```
[3] FD->bk = BK;  
[4] BK->fd = FD;  
}
```

练习 14.7 在 UNIX 系统中配置一个网络服务器的时候，如果要应用最小特权策略，你怎样在软件部件上设置所有权和访问许可？

练习 14.8 据说把软件的漏洞全部暴露出来可以提升软件的安全性。当发现一个漏洞的时候，应该采取那些措施来使得问题最终在用户端能够得到顺利解决呢？一步步分系讨论到底应该考虑那些问题呢？全部揭露出来有什么优点和缺点呢？

第 15 章 新的访问控制范例

随着因特网和万维网的发展，越来越多的根本没有安全意识的用户直接开始使用新的信息技术产品。移动代码通过因特网传播，在客户端上运行。电子商务给全球带来了新的商机。我们将再次面临当前正使用的信息技术系统的巨大改变。我们不得不去思考下面的问题：当前的安全范例仍然适合吗，是否需要引入新的策略和新的执行机制。

在编写本书的此刻，这个过渡阶段还没有结束。一些新的有关新环境下的安全策略的概念已经出现，但还需要更多的经验积累才能给出明确的建议。

目标

- 开发新的访问控制范例。
- 分析基于代码的访问控制范例的背景和基本原理。
- 介绍主要的安全执行算法——堆栈游走，用于基于代码的访问控制中。
- 介绍 Java 安全模型和 .NET 安全框架。

15.1 引言

在万维网出现之前，分布运算的主要案例都采用客户端—服务器模式体系结构，即客户端（希望）在服务端上执行计算，服务端则通过对客户端的认证来保护自己。Kerberos 协议就是针对这种环境而设计一个重要的认证服务（参考 12.4 节）。然而这种体系结构在一些核心方面（领域）已经发生改变，计算已经从服务端转移到客户端。比如浏览网页时，嵌入在网页中的程序（applet）将由客户端浏览器执行。访问操作的本质已经发生改变，程序不再是简单的向操作系统或者数据库发送读写请求，而是被发送到另外一端去执行。服务器还执行来自客户端的脚本，关于脚本方面的安全已在 14.5.2 节中讨论。客户端从服务端接受 applet 并且可能在 cookie 中保存会话状态信息（参考 15.5 节）。那么，万维网到底发生了哪些改变呢？

- 程序和数据的分界线已经变得模糊，通过将可执行上下文（applet）嵌入文档中来创建可以处理用户输入的交互式网页。
- 计算已经转移到客户端。将可执行代码嵌入文档中，让它在更有效的客户端上执行，这样服务器就把计算任务转交给了客户端从而释放出自己的资源。因此，客户端需要保护自己以防止来自可能的虚假内容提供者的威胁。
- 因为计算转移到了客户端，客户端将被迫自己决定并执行安全策略。用户的角色被转换成管理员和安全策略的制定者。
- 浏览器成了可信计算基（TCB）的一部分。

Web 也已经创造了一个软件分发的新范例，软件变成了一种可以从因特网下载的内容。如今，特别是升级和补丁都广泛采用这种分发方式。但有一个以前的经验教训值得我们学习。在个人电脑发展的初期，软件的分发是通过软盘进行的，接受不明来源的软盘成为传播计算机病毒的主要途径。许多组织意识到必须适宜地管制重要的软盘，在万维网上也有这样的风险，类似的保护措施也必须实施。

更重要的一方面，我们必须正视那些流窜于机器之间以从地方的不同收集信息或者是寻找

空闲运算资源的移动代码。客户端需要保护自己来防止来自移动代码的威胁，同时移动代码也需要运行它的机器提供保护。

尽管万维网还没有带来严重的安全问题，但是它已经将需要实施的内容改变到这样一种程度，以至我们不得不重新考虑新的访问控制范例。本章将探讨在访问控制上的变化。

15.1.1 访问控制范例的改变

在 4.2 节中的访问控制模型中，主角发送需要被认证的访问请求，安全策略授权主角访问一个对象。但是这里存在一个严重的潜在假想，即主角只和知道的人通信。Unix 和 Windows 下基于身份的访问控制都是在这个模型上建立起来的，并实现了基于身份的访问控制(identity-based access control, IBAC)。这种模型最初是由大学和实验室这类组织创建的。这种访问控制方式建立在一些重要的组织假设基础上，如组织可以为其成员授权，成员(用户)可以被定位到物理地址，审计日志记录了用户的可能被追究的行为。在这样的背景下，安全策略自然就是指用户的身份，访问控制就是规定用户的身份可验证。生物测定技术(3.7 节)作为一个逻辑步骤，正朝更强的基于身份的访问控制迈进。

这些有关组织的假设和访问控制的部署有着密切的联系。访问规则是本地的，没有必要去寻找适合当前请求的规则或者是寻找访问主体必须提供的凭证。所有的规则和对对象都被保存为访问控制列表(ACL)。凭证就是访问主体所在的 UID(Windows 中的 SID)。规则的实施统一执行，引用监控器在做决定时并不与其他参与方协商。许可用于常用的简单操作，如读、写、执行。在设置安全策略的时候，还需要考虑一致性(homogeneity)的程度。同一组织定义有组织的和自动的安全策略。

15.1.2 修订的有关访问控制的术语

由于历史的原因，当前使用的有关访问控制的术语都和基于身份的访问控制相关。下面三节的主题是基于代码的访问控制，我们将不深究“是谁做的这个声明”，因此我们讨论的不是在没有认证的情况下进行访问控制，就是重新解释认证并扩大它的含义，以便它能支持外部提交的和请求关联的证据的验证(图 15-1)。额外地，我们可能需要把本地证据和一个请求联系起来。比如，请求可能只是当前会话的一部分或者在决策是否授权时可能会依赖当前的日期和时间，或许我们还需要让认证包含这个关联的验证。

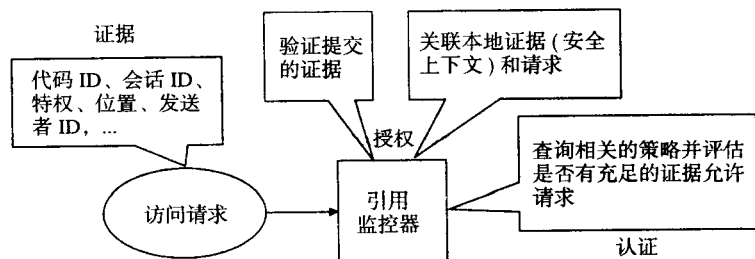


图 15-1 开放环境下的访问控制

其次，我们不得不去寻找适合这个请求的策略，然后检查要承认这个请求其所持的认证证据是否充足。在一个使用了许可的系统中，我们可以检查所有的请求许可(required permission)是否都在授权的许可中。这个短语可以称为授权(authorization)，它并不表示谁被允许去访问一个资源，而是要获得一个资源的访问许可必须提供的凭证。

为了简化描述,图 15-1 中的引用监控器掌管了认证和授权的所有方面。但在实际中这些功能却是由不同的部件来实施的。

- 策略管理点(PAP)制定策略。
- 策略决定点(PDP)评估适用的策略并且决策授权。
- 策略执行点(PEP)执行访问控制,它将策略抉择发送到 PDP 然后执行接受的被授权策略。
- 策略信息点(PIP)提供证据来源。

15.2 基于代码的访问控制

因特网让从未谋面的我们走到了一起。在应用中,例如电子商务,允许未知的实体访问我们的系统通常都是必须的。自然地,这就会潜在地危及我们的安全,因为我们已经从一个封闭的环境走向另外一个开放的环境。

- 我们和从未认识的人打交道,在我们的访问控制规则中难以获取他们的身份。
- 我们无法知道对方在现实中的地址。
- 在其他的一些方面,我们可能没有实际的权力。即使我们可以证明有些人应当为某些行为负责,我们可能仍然无法问责(追究)他们。因为他们可能生活在不同的司法社会下,导致诉诸法律过程可能会很缓慢,或者多余,或者是代价相对较大,这些都是制定安全策略时不得不考虑的问题。与消费者的关系和与被雇用者的关系在很多方面都不一样。

在上述环境下,设置安全策略时用户身份就显得不是那么有用。因此,一个请求的来源就不再是访问控制的有效参数。同时,引用监控器不再是接受读、写、执行请求,而是一个完整的程序(applet 或者脚本)。访问控制需要决定是否运行 applet 以及给予将要执行的 applet 什么权限。Java 沙盒用基于代码访问控制方法来代替基于用户身份的访问控制。在该访问控制的策略中使用如下的证据。

- **代码来源:**代码是来自本地还是来自远程?如果是来自远程的代码,那它是从哪个 URL 下载的呢?我们可以为这个 URL 参数设置一个区域,然后在其基础上定义安全策略。一个简单的示例是,我们可以将策略简单的定义为代码是来自内联网还是因特网。
- **代码签名:**代码是否被一个可信任的提供者进行了数字签名?在这里,信任只是简单地表明:我们已经许可了执行来自指定提供者的代码并给予一定权限。通常这只是一个操作决策,而不是对某个作者的可信赖度的决策。
- **代码标识:**有时候我们可能决定给予可信程序一些特权,家庭银行应用程序就是这样的一种示例。计算接受到的 applet 的 hash 值,然后检查其是否在可信的 applet 列表上。同样,这更多是一个操作决定,而不是基于技术上的对 applet 可信赖度的断定。
- **代码校验:**是否运行一个 applet 的决定基于代码的某些特殊属性,比如它可能写的文件或者是它可能要连接的远程地址。创建一个符合这类安全属性监控的证据不是一项简单的任务。因此,需要作者提供校验,在访问控制在做决定的时候,引用监控器将检查其校验值。研究人员提出了携带校验的代码的理念。判定有用的安全属性实际上要比检验它们是否拥有该属性难得多。

我们将这种访问控制称为基于代码的访问控制,同时也可称之为基于证据的访问控制(evidence-based access control)。为了使安全策略可管理,许可通常都被赋值给代码环境(.NET 术语集),个别的软件部件从包含它们的代码环境中继承许可。

15.2.1 堆栈检查

主体代表主角行动。在基于身份的访问控制中,用户是主角。用户登录时,一个进程被创建

并运行在该用户的 UID 下, 该进程的访问权许可从它的 UID 继承而来。通常, 在执行这个进程的过程中, 许可不会改变。当需要改变的时候, 新的进程产生后将运行在不同于父进程的 UID 下。UNIX 下的 SUID 程序就是这样的一个典型的例子。

在基于代码的访问控制中, 一个软件部件调用另外一个部件时, 执行这些部件的进程的有效许可将发生改变。判断有效许可的算法必须考虑调用部件和被调用部件的许可, 因此我们不得不决定一个可信部件调用不可信部件时的许可, 反之当不可信部件调用可信部件时, 我们也必须作出决定。有如下示例, 函数 g 有访问资源 R 的许可, 但是函数 f 却没有。当函数 g 调用函数 f 而 f 又去请求资源 R (图 15-2a)。可以授权给这个访问吗? 这种情况下保守的策略是拒绝, 因为函数 f 自身不具备对资源 R 的访问许可。但是 g 可以明确授予 f 访问 R 的许可。相反的, 让 f 调用 g , g 再去访问资源 R (图 15-2b)。这个访问应当被授权吗? 保守的回答还是拒绝。因为 g 可能已经被 f 使用诱饵式攻击 (luring attack; 也被称为困惑的代理人攻击, 参见 5.3.5 节) 利用 (却不知道), 但是 g 却断言这个访问权限。在上面的两个示例中, 我们在做访问决策的时候都需要知道整个访问链。

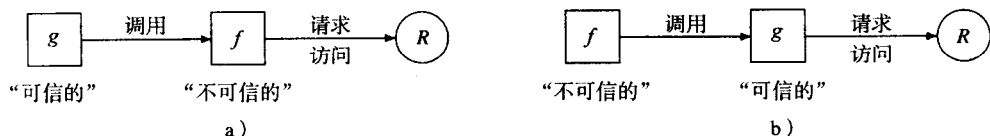


图 15-2 基于代码的访问控制中的调用链

习惯上, 我们将那些具有高许可的部件称为可信部件。但是这并不代表有理由让我们相信这样的部件不需要安全策略的约束, 它仅仅代表这类部件的执行需要像系统级别这样的许可而已。同样, 不可信部件并不是不能执行的部件, 而只是给予其相对较少的许可而已。

Java 虚拟机和 .NET 通用语言运行时都使用调用栈去管理代码的执行。每发生一个函数调用就在栈上创建一幅新帧, 每一幅帧都包含函数的本地状态, 比如直接授予给它的许可。为了计算有效的许可, 到目前惰性计算 (lazy evaluation) 仍然为软件开发

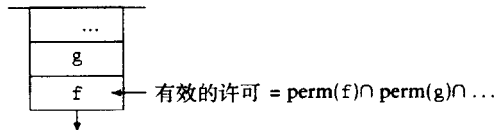


图 15-3 堆栈游走中的有效许可计算

者所青睐。当函数请求访问一个受保护的资源, 堆栈遍历 (stack walk) 用来确定调用者是否有要求的许可。最终, 调用者有效许可的计算取决于调用栈上所有函数的许可的交集 (图 15-3)。

如果这些策略被严格的执行, 受控调用的选项将受到严重的约束。因此, 通常提供者都提供了选项用于代码的许可断言 (assert)。许可断言附着在当前栈帧上, 当从调用部件返回时撤销。堆栈遍历在遇到许可断言的帧并授予这个许可时终止 (至此所有被检查的帧都具有这个许可)。如果还有要求的许可没有被断言, 堆栈遍历将继续执行下去。断言允许不可信的调用者调用可信的方法。

编写部件并给其赋予许可的程序员必须考虑到不可信部件可能使用他们实现的函数因而必须建立适当的防御策略。同时, 他们也必须意识到由于调用者许可不够, 代码在运行时原有的许可可能会丢失。

15.2.2 基于历史的访问控制

堆栈审查通过重用调用栈去尝试获得该栈的原始使用方法。就性能来看, 出于安全的因素, 通常的优化或许不得不失效。实际上通过堆栈审查来精确地确定安全保证是很困难的, 另外和

安全相关地参数不一定就放在堆栈上。因此执行这种机制地安全策略很难适应高级别地安全策略。

我们将使用尾调用清除(tail call elimination)去说明第一点, 函数最后一个函数调用指令被称为尾调用。考虑如下的示例:

```
void g ( )           void f ( )
{
    ...;
    f ( ); //tail call
    return;
}
{
    ...;
}
```

一旦 $f()$ 被调用, g 的帧就不再需要了,它被 f 的新帧覆盖掉。事实上, f 直接返回到 g 的调用者。在递归函数中这样的优化特别的有效。使用尾调用清除,不可信的调用者可能在堆栈上不留下任何踪迹。在我们的例子中,让 f 发起一个需要许可 p 的访问,但是许可 p 并没有授予给 g 。如果不使用尾调用清除,堆栈遍历在检查 g 的栈帧时将会产生异常。然而使用尾调用清除,堆栈遍历将继续持续下去,这可能导致一个应当被禁止的访问却被许可(图 15-4a 和图 15-4b)。

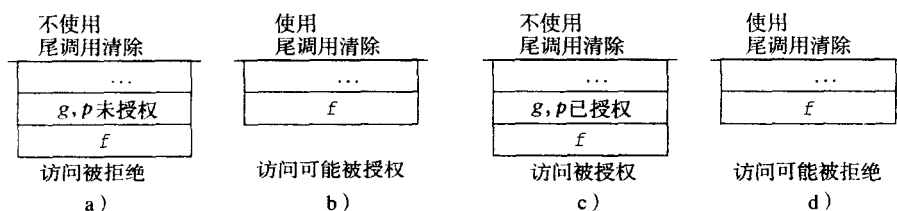


图 15-4 尾调用清除的影响

另外一方面,一个可信的调用者可能取消一个断言的许可。在上面的示例中,让 f 有许可 p , g 断言许可 p ,并假定先前的调用者不具有这个许可。不使用尾调用清除,堆栈遍历将停在 g 的栈帧上并且进行授权。而使用尾调用清除,堆栈遍历由于看不到断言就会禁止该访问。在这里,一个应当被授权的访问却被拒绝了(图 15-4c 和图 15-4d)。

为了避免堆栈审查安全机制的缺点,放弃惰性计算的措施而采用热情计算(eager evaluation)去前瞻性地估算调用者的许可。在基于历史的访问控制中,在加载的时候静态许可 S 被关联到代码的每一部分上,和每一个执行单元相关联的当前许可 D 在规则 $D = D \cap S$ 下自动更新。

15.3 Java 安全

为了在 Web 上畅快地冲浪,用户必须准备好接受来自任何吸引他们的站点的可执行上下文(applet)。这要求用户能够控制 applet 的行为。applet 是由 Web 浏览器中的虚拟机解释执行的程序。这样就要求虚拟机去执行访问控制,Java 使用沙盒(sandbox)限制 applet,同时 applet 使用的语言也使得其很难造成损害。为了正确有效地使用执行环境的访问控制机制,安全策略必须正确的设置。

15.3.1 执行模型

Java 是安全的面向对象的语言。Java 的源代码被编译成独立于机器的字节码,并以类文件的方式存储。Java 字节码和汇编语言很相似。特定于平台的虚拟机将解释这些字节码,并将其翻译成特定于平台机器的指令。执行一个程序时,类加载器会加载所有被要求的附加类。图 15-5 给出了使用 Java applet 的典型概况。applet 使用 Java 语言编写,然后被编译成字节码放在服务器

上, 最终被客户机上的浏览器执行, 支持 Java 的浏览器有自己的 Java 虚拟机。Java 虚拟机有 3 个安全部件: 字节码校验器、类加载器和安全管理器。目前的浏览器都倾向于执行如下这类可选策略, 尽管它们并不是必需的。关于 applet 的典型安全策略如下:

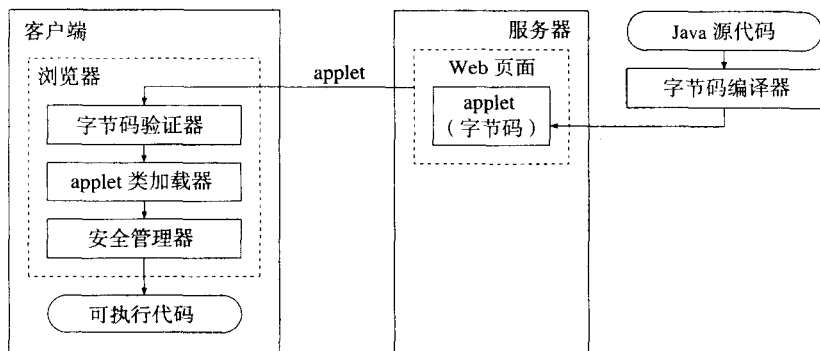


图 15-5 Java 执行模型

- applet 不能访问用户的文件系统。
- applet 不能获得像用户名、电子邮件地址、机器配置这类的信息。
- applet 只能回连到它们来自的服务器——如果它需要远程连接。
- applet 只能弹出标志着“不可信的”窗口。
- applet 不能通过利用系统类替换自己的类来重新配置 Java 虚拟机。

15.3.2 Java 1 安全模型

最初的 Java 安全模型只实现了很简单的安全策略(图 15-6)。相对本地代码而言, 未标识的 applet 被严格地限制在沙盒中。从 Java 1.1 版本开始, 标识的代码不再被严格限制。Java 基本的安全策略不提供任何中间的控制, 这大大地降低了给“半可信”applet(例如, 作为家庭银行应用一部分的 applet)额外许可的灵活性。

另外, 代码来自本地还是远程并不是一个精确的安全标识。本地文件系统的某些部分可能是存在于其他机器上的, 这样它们将被沙盒严格限制。相反地, 下载的软件一旦被缓存或者被安装在本地系统上就变成可信的代码。为了实现更灵活的安全策略, 需要一个可定制的安全管理器, 这是一项需要安全技术和软件设计技术的重大任务。

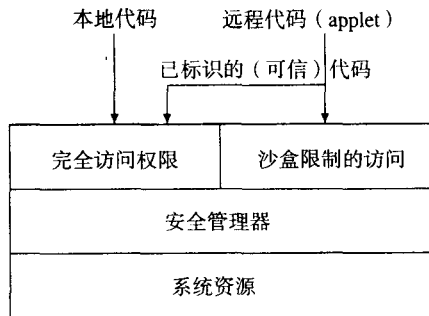


图 15-6 Java 1 的安全模型

15.3.3 Java 2 安全模型

Java 2 安全模型将安全策略的规范和策略的执行分开, 为基于代码的访问控制提供了一个灵活的框架, 同时简化了一些策略的执行过程。安全策略通过映射属性描述码到授权的许可的这种方式来授权代码对资源的访问许可。安全管理器在请求访问受保护的资源时被调用, 以执行安全策略, 它收集相关的证据然后激活访问控制, 策略决定点做最终的决定。

集中考虑常见功能时, 我们可以看到如下的 Java 2 整体安全模型。这个模型包括四大部分:

语言安全(字节码验证器),安全扩展系统(类加载器),策略规范者和策略实施者。如果想获得这种模型的更多细节信息可参考 Gong, Dageforde and Ellison(2003)的著作。

15.3.4 字节码校验器

第一道防线是 applet 所使用的编程语言,在很大程度上语言能够限制编写出恶意的 applet。Java applet 是用字节码编写,它自身具有一些安全属性。字节码校验器(Byte Code Verifier)分析 Java 类文件,执行语法检查,使用定理证明程序和数据流分析去检查静态类型,以确保在 applet 执行前这些异常都被检查到。校验器保证如下的属性:

- 类文件是正确的格式。
- 堆栈不会溢出。
- 调用都使用了正确类型的参数并且都返回了正确的类型。
- 没有非法的数据类型转换。
- 所有对其他类的引用操作都是合法的。
- 没有访问违规操作,私有区域对对象外部不可见。

字节码校验器减轻了解释程序的工作量,由于校验器保证了代码的(属性)可靠性,所以在运行时不再需要检查。字节码验证无法涵盖所有的安全问题,因此安全还必须依赖运行环境的安全机制。有关字节码校验精确形式描述的实验还再继续着,最终目标是找到证据证明字节码校验能够提供他们所宣称的保证。

15.3.5 类加载器

Java 平台是一个可扩展的系统(extensible system)。类仅在链接需要的时候才加载(惰性加载)。为了减少物理内存的使用和提升系统的响应时间,类加载总是尽量最大延迟。共有三种类加载器。其中,引导类加载器(Bootstrap Class Loader)加载组成虚拟机核心的系统类,这种方式加载的类对系统资源的访问(在编写时)拥有相对宽松的限制。引导类加载器是和平台相关的,通常使用底层语言如 C 编写,并且通常从本地文件系统启动。扩展类加载器加载安装的可选软件包中的类。应用程序类加载器(Application Class Loader)加载用户定义类。

类加载器加载类时,它读取字节码文件,定义类并赋予它一定的保护域(参见 15.3.6 节)。Java 虚拟机在链接时执行检查确保类型安全,和运行时检查相比,链接时检查具有只需要检查一次的优点。每一个 Java 虚拟机都有类文件校验器去保证不可信的类是安全的。类文件校验检查 Java 语言的约束是否得到执行和类文件是否有正确的内部结构。类文件验证调用字节码校验器。

每一个类都由它的类型和定义它的类加载器唯一定义。因此,类加载器为不同的类提供了单独的名字空间。浏览器加载 applet 的时候从单独的类加载器中加载各种资源。这些 applet 可能包含相同名字的类,但是这些类却被 Java 虚拟机识别为需引起注意的类型,同时支持用户自定义的类加载规则。比如,用户可定义类加载器可以做到怎么去寻找类或者标识那些从给定资源中加载的类的安全属性。

15.3.6 策略

安全策略将代码属性映射给许可。许可代表了系统资源和对这些资源允许的操作。许可的格式有(名字、操作)等,比如(/tmp/abc, read)表示对文件的只读许可。许可被授予给代码(授权许可),访问资源的时候同样需要许可。Java 只使用积极的许可,开发者并不被局限于使用已经定义的许可,它们可以定义特定的应用程序的许可。

有一个许可类的分层结构。不同许可间的关系由 `implies` 和 `equals` 方法定义。`AllPermission` 是一张具有全部许可的通行证, 针对系统管理员这样的需要大量许可才能完成他们的工作的实体, 它被设计成流线型策略运算。`unresolvedPermission` 是那些尚待解决的许可占位等, 比如, 也许存在 `Policy` 对象创建后某些许可类尚未加载的情况。为了支持策略管理, 许可应当能被许可集 (`permission set`) 所枚举。

和代码相关的安全参数是源代码 (`code source`), 包括 `URL` (代码起源) 和可能的数字证明 (代码签署者) 以及代表用户或者服务的主角。保护域是一种管理安全的手段, 每一个类在加载的时候就和保护域相关了, 保护域包括源代码, 主角, 类加载器的引用, 授予代码的静态许可和代码加载时被标识的许可。保护域是一个不可改变 (`immutable`) 的对象, 一旦它们被创建就不允许改变。

策略存放在策略文件中。策略文件可能包括最多一个主存储表项和任意的授权表项。主存储表项标识了到哪里去寻找在授权表项中给出的签署者的公钥。授权表项对代码基以及签署者和主角列表分配许可这 3 个域都是可选的。有可能赋予签署者单个的许可。如果这样, 则只有从指定的签署者签署的文件中找到的许可才是有效的。

15.3.7 安全管理器

Java 虚拟机中的引用监控器就是安全管理器, 在进程请求访问受保护的资源时被调用来检查进程是否拥有所要求的许可。程序通过调用安全管理器的 `checkPermission` 方法去请求这个检查。该方法可以作为唯一参数与请求的许可一起调用, 也可能与请求的许可和执行上下文一起调用。执行环境捕获了执行堆栈上的内容。安全管理器的 `checkPermission` 方法依次调用访问控制器的 `checkPermission` 方法。

访问控制器对所有许可执行统一的访问决策算法。为了计算授权的许可, 访问控制器检查提交的执行上下文, 即执行堆栈, 并运行堆栈游走。栈上的每一个方法都对应一个类, 而每一个类又属于一个保护域。保护域定义授权给该方法的许可。最基本的访问控制算法就是计算执行上下文中所有方法已获得的许可的交集。如果堆栈中的所有方法均被授予了该所需许可, 则访问控制器同意该访问这个算法现在又被扩展了, 使得方法在忽略先前调用者时可以通过调用 `doPrivileged` 方法来断言许可, 从而结束堆栈遍历。进一步的改进算法可用于处理与方法继承和可执行上下文继承相关的内容。

15.3.8 小结

Java 2 安全模型使用的方法被称为“称述的安全”。应用程序的作者通过在代码中加入相关的检查来断言访问受保护的资源需要哪些许可。这使得定义特定应用程序许可具备充分的灵活性。部署应用程序的管理员通过具体指定安全策略来分配他们的许可。访问控制器提供了共享的唯一访问控制算法, 以确保安全策略实施的一致性。应用程序编写者不需要实现他们个人的、特定应用程序的访问控制逻辑。

Web 浏览器中的 Java 虚拟机在操作系统之上的层执行安全机制。一旦用户访问位于这个安全机制下的层时, 比如说访问其他应用程序而不是浏览器, 所有的有关系统完整性的保护都将失效。此外, Java 平台不是坚固的, 因为 Java 仅仅是被设计为一类安全的语言。随着时间的推移, 一些安全问题被发现并得到了解决。第 14 章将介绍一些与类型安全相关的示例。大多数情况, 攻击者是通过破坏类型系统进入的。这将是给那些认为仅仅只需要面向对象抽象机制而不需要付出额外的努力解决对象管理系统下的安全就可以构建一个高强度的安全的人一个严重警

告。构建一个复杂而严密的系统是一项艰巨的任务。与 Java 安全相关的工作仍将继续下去。

15.4 .NET 安全框架

本节将给出 .NET 安全特性的概述并介绍它的一些术语。*La Macchia et al. (2002)* 更详细地介绍了 .NET 安全。

15.4.1 通用语言运行库

通用语言运行库 (CLR) 是一个支持多种编程语言如 C#、VB、VJ#，托管 C++ 等的运行时系统。通用语言规范 (CLS) 规定了 .NET 所有语言的通用需求。C# 是主要的语言，它是一种类型安全的编程语言。在某种程度上，C# 语言的设计是建立在从 Java 语言使用中获取的经验之上的。用 .NET 的任何一种语言编写的代码都被编译成微软中间语言 (Microsoft Intermediate Language, MSIL) 代码。MSIL 与 Java 字节代码的概念相似。在 MSIL 代码执行前，这些代码被送到实时 (just in time, JIT) 编译器。公共语言库加载并运行代码，执行安全检查，自动进行垃圾回收以便编程者不必明确释放或者删除他们不再需要的对象。从构建上讲，公共语言库和 Java 虚拟机相当。

托管代码 (managed code)，被编译成在 .NET 框架下运行并受公共语言库控制的代码。本地代码 (native code)，又名非托管代码，即被编译成特定硬件平台的机器语言代码。本地代码不受公共语言库的控制，在公共语言库下面一层工作。托管代码到本地代码的调用是安全的关键，因为此时任何由公共语言库提供的保证都不再实用了。目前，特定的措施已经被用于检查这样一个调用会不会导致安全违规。

程序集 (assembly) 是微软中间语言代码的逻辑单元，它包含的元数据 (metadata) 提供了如下信息：程序集的全称、引用的程序集、可见性、继承的类、每一个定义的类执行的接口，以及类成员的信息 (如方法、域和属性)。

15.4.2 基于代码身份的安全

.NET 平台使用基于代码的访问控制，即由 *La Macchia et al. (2002)* 提出的基于代码身份的安全。基本的思想是为代码指定不同的可信度，精确地说就是赋予代码不同的许可，使用最小特权法则是一个很好的原则。调用那些完全可信的拥有所有许可的代码，不可信的只有非常有限的许可代码，以及那些处于部分受信间的许可的代码混合了访问控制的可操作方面，并说明了为什么许可可以被以一定的方式赋予。

基于代码身份的安全引用了代码的身份，安全策略引用关于程序集的证据，授权代码而不是用户来访问资源。在代码运行中证据被动态地收集。一些证据项通常是无法预先知道的，如源程序集的 URL，代码身份的认证用来收集和校验程序集证据的过程。

15.4.3 证据

程序集的证据可以包含默认类中的对象，如程序集被加载的源现场，源 URL，程序集的 hash 值，代码的身份签名，强名称签名和安全区域。当然也可以使用任何其他对象来作为定义特定程序的访问控制证据，但是这需要添加自定义代码去处理这些证据。许可请求证据确定了程序集运行必须具有的许可，包含其可能已经获得的许可或者绝对不能获得的许可。

证据由两种方式提供。由装入程序集的实体 (称为宿主) 提供，或者由初始化 CLR 的非托管实体 (比如 IE) 来提供，或者装载其他托管代码的托管代码。这样的证据称为宿主提供的 (host-

provided) 证据, 它使用前面提及到的默认类。程序集提供的 (assembly-provided) 证据由程序集自己使用应用程序特定的类提供, 它不能覆盖宿主提供的证据。证据和程序集以及所谓的应用程序域相关联。应用程序域包含加载的程序集, 它提供执行代码的进程内隔离。所有的 .NET 代码都必须运行在一个应用程序域中。

15.4.4 强名称

强名称为来自不同发行者的程序集创建单独的保护名称空间。发行者的公钥伴随着程序集名的其他部分被添加到元数据中, 程序集在编译的时候进行数字签名, 签名最后被写入到程序集中。这个数字签名以后可以被 .NET 框架校验, 从而保护名称空间中的程序集不被修改或欺骗。和认证代码签名相比, 在这个处理过程中不需要任何的证书。

15.4.5 许可

代码访问许可 (code access permission) 代表访问资源或者是执行受保护的操作的许可, 比如访问非托管代码。.NET 框架提供如下的这类代码访问许可, `FileIOPermission` 提供对文件或者目录的读、写、添加的操作许可, `EventLogPermission` 提供访问事件日志服务的读写许可, `PrintingPermission` 提供访问打印机的许可, `SecurityPermission` 提供断言许可、访问非托管代码、跳过校验的许可。事实上, 有很多这类的内置的许可。

身份许可 (identity permission) 代表代码身份的证据, 这样一来, 引用这些特定的代码身份的策略能够被定义和加强。身份许可和程序集的发行者、强名称、源位置、源 URL 或者源区域相关。`PrincipalPermission` 代表了用户的身份, 它不同于代码访问许可, 用于基于身份的安全策略中。为了简化策略管理, 许可可以从许可集中获得。

许可由安全策略授权给程序集, 必选的许可通过在代码中设置安全请求 (security demand) 来标识。在执行中, 安全请求触发堆栈遍历去检查许可是否已经授权。陈述的要求存储在程序集的元数据内, 通过查看元数据可以轻易地审查。检查发生在方法的开始处, 实时安全操作只能按照陈述的格式解析。强制的 (imperative) 请求设置在代码中, 它使得更多复杂的安全逻辑在访问请求时也能够处理动态参数。

15.4.6 安全策略

安全策略将证据翻译成许可, 代码组是构建这个映射的主要单元。一个代码组包括成员状态和策略声明。策略陈述本质上就是成员所持有的许可。在装载的时候, 成员状态检查程序集提供的证据是否符合给定的参数。例如, 可以对比区域证据和存储在成员状态中的区域, 或者对比装载的程序集的 hash 值和成员状态中的哈希值。

为了支持策略管理, 代码组按层级编排。许可从代码组根开始分解, 寻找匹配的子代码组。对排他性代码组 (exclusive code group) 匹配而言, 这样的代码组不需要和其他代码组一起评估, 它只能从这个代码组得到许可。更多的, 有 4 种策略级别: 组、机器、用户、应用程序域。策略级别包含一个已命名的许可集, 一个代码组层次, 一个完全可信程序集列表。证据在每一个级别上单独的评估, 结果为交集。

15.4.7 堆栈遍历

一次请求安全行为会触发一次堆栈游走。在基本的模型中, 调用堆栈上的所有程序集都会被检查 (发出命令的当前代码除外), 且为了能成功访问, 要求其有请求的许可。这个算法可以

被 Assert(断言)操作所修改,断言操作可以断言许可并结束堆栈游走。Deny(拒绝)操作也可以通过抛出安全异常终止堆栈游走。该操作在测试中十分重要。只允许行为和拒绝除了特定许可的其他所有的许可的 Deny 操作等价。

15.4.8 小结

.NET 使用和 Java 类似的安全策略,托管代码使用类似 C# 的类型安全语言,公共语言运行库验证中间语言(IL)代码来确保类型安全。代码而非用户被授权去访问资源,并且代码执行时必须要通过认证。安全执行算法使用堆栈遍历。构建安全策略有很多可用的方法,解释要求的许可有两种不同的方式。上面提及到的 .NET 框架都提供支持。在实际应用中,你需要运用谋略去设置策略,并给程序集赋予许可。在编写时,还会遇到更多的新问题。

15.5 cookie

现在,我们放下基于代码的访问控制去看看访问控制中的其他范例的改变。接下来讨论万维网会话管理的安全方面,作为万维网骨干的 http 协议最初被设计用来传输 HTML 文档文件,它是一个无状态的协议。所有的 http 请求都被当成一个单独的事件,即使它们来自同一个客户端。因此,一些相关的操作任务不得不经常地重复。比如访问一个需要密码的网页,每次点击这个网页时都需要把密码返回给对方。http 1.0 使用了会话机制来解决这个问题,浏览器保存第一次请求这个页面时候输入的密码,这样在后续响应服务器时便可自动加入这个密码。

处理包含多步事务的应用程序(比如网上购物)要求保持这个事务的状态,以便在事务过程中发生中断时,能够使服务端和客户端之间保持一致的状态。为了实现这个目标,客户端的浏览器将事务的状态保存为 cookie(图 15-7),这样服务端通过获得这个 cookies 去得到事务的客户端的当前状态。利用 cookie,可以创建安全的 http 会话。

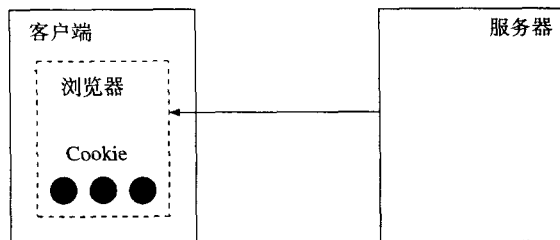


图 15-7 保存在客户端的 cookie

根据保存的 cookie 所持续的时间,会话的概念已超出了一个单一事务的概念。服务提供者可以利用 cookie 来管理客户的关系,客户通过 http 请求和服务提供者联系,但是 http 请求却没有必要去识别单个的用户。因此服务端可能要求浏览器保存永久的 cookie,这个 cookie 保存了客户下次调用时服务器需要引用的信息。

cookie 会带来安全问题吗? cookie 不可能侵害系统的完整性。它们仅仅是数据而不是可执行代码,独立的 cookie 不会向服务端透露信息。毕竟是服务端要求浏览器保存 cookie 的。通常 cookie 只是一些特殊的域,服务器只将访问权给予属于它的域的 cookie。在这种情况下,也不存在新的机密性问题。然而我们不得不去考虑应用级的攻击。攻击者可能是一个服务器,通过创建客户配置文件,结合由不同服务器放置的 cookie 或者是长时间观察客户行为得到的信息侵犯客户的隐私。攻击者也可能是一个客户端,它试图通过改变 cookie 来从服务器得到本来没有授权给客户的好处。例如,当服务器使用 cookie 保存信用体制中的红利点时,客户端可能通过增加

积分来获得更大的折扣。这种攻击称为 cookie 毒药攻击。攻击者可能是第三方，它通过学习性的过程来猜测客户的 cookie。然后利用虚假的 cookie 去扮演合法的客户。

客户端可以通过设置浏览器来控制 cookie 的存储以保护它们自己，在存储 cookie 前，浏览器可以请求许可。这很容易变成一件烦人的事或者很容易致使 cookie 被一同阻止并且在会话结束时删除 cookie 的选择权。服务器可以通过加密 cookie 来保护自己，使用适当的认证也能防治欺骗攻击。

Hogben, Jackson, Woliken(2002) 提出了一个有趣合法的 cookie 难题——P3P，即 Platform for Privacy Preference。P3P 使得 Web 站点可以解释收集到的用标准的 XML 格式语言描述的数据。在早期的 P3P 版本中，只有关于重获 cookie 的策略才会被执行。从技术角度看这是合理的，但是却违反了欧盟个人资料保护指令。指令要求在写入个人数据时需要请求用户的允许，指令关注与保存在个人资料数据库中相关的隐私。当用户相关的数据要保存在其他的系统上时，在写数据时需要请求允许。cookie 在用户的机器上保存属于用户的信息。这个敏感的操作对于其他部分只允许读。

经验教训

立法可能包含老的技术，IT 相关的法律面临着它们被制定的那个时代的技术挑战。同时对于当时那些适用于特定应用的技术，立法者也可能混合一些关于带有先见之明的设想。因此一项法律不仅仅是规定要保护的目标（尽管该项到目前都没有改变），还有保护的机制，尽管这些机制在新的应用中不一定是最好的选择。

15.6 简单公钥基础设施

在旧的访问控制范例中，访问规则存储在本地受保护的存储器中。在分散式的访问控制系统里，我们可以通过加密的方法来保护访问规则的完整性。特别是我们可以使用数字签名证书来编制规则。基于代码的访问控制可以用 X.509 身份证书和 12.5.3 中讨论的属性证书来实现。

简单公钥基础设施(Simple Public Key Infrastructure; SPKI RFC 2692, 2693)，是一种支持无用户身份的授权方案。SPKI 的设计者们假定访问控制中使用的名字仅仅是个本地含义，访问许可是来自其他属性而非用户名字。因此，在这种访问控制下，名字仅仅是访问许可(授权)的指针。安全策略在给定的域中设置规则，比如说大学系别。用来表示策略的名字必须和域一起才有意义，不再要求名字是全局唯一的。

当域间有交叠时，我们需要从其他的本地名称空间引用名称。为了避免名称空间的混乱，我们需要给名称空间一个全局的唯一的标识。公钥加密系统提供了很好的解决这个问题的方法。随机生成的公钥/私钥对在极大程度上是唯一的，公钥的 hash 值也是如此。发行者的公钥(或公钥的 hash 值)作为其定义的名称空间的唯一标识。在 SPKI 术语中，这样的密钥(key)被称为主角(principal)^①。使用私钥签名的名称证书定义本地名称空间中的名字。

通过授权证书访问许可和公钥直接联系了起来，授权证书至少包含发行者和主体，还可能包含授权比特位，许可和有效条件。发行者签署证书并授权主体，也就是授予主体访问许可。因此，发行者是批准源，策略也由其设置。典型的主体就是公钥，或者密钥的哈希值，抑或是密钥的名称。证书校验链的根密钥存储在访问控制列表(ACL)中。身份证书将用户的名字绑定到一个密钥，它由 CA 发行而不是授权证书者发行。参见图 15-8。

^① 这里的术语主角、主体和第 4 章中的意义不同。

访问权限被赋给公钥，相应的私钥持有者可以通过发行对应许可的证书将许可授权给其他主体，有两种方法来严格这个授权。授权者只能授予它们所有的许可，假设要观察一个这样的设计决策：一个 *key* 持有者可能会授权一个根本无法执行的许可，它需要生成一个新的密钥对然后把许可授予给那个 *key*。在这种情况下，这样的设计决策被证明是正确的。另外一种方式是用一个标志指示该证书的许可是否可以再进一步授权给其他人。

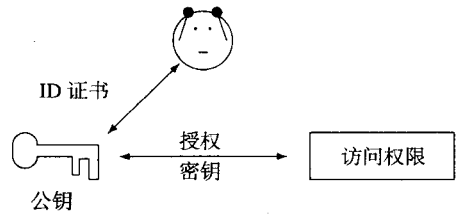


图 15-8 SPKI 中的认证

授权证书和访问控制列表(ACL)有相同的功能，仅仅是在保护的模式上不同而已。证书是由发行者签名的，而访问控制列表没有签名也没有发行者，并且它们仅仅存储在本地从不被传输。证书和访问控制列表项的抽象符号为元组。授权处理的算法由一个 5 元组描述：<发行者(issuer)，主体(subject)，委派(delegation)，授权(authorization)，有效期(validity date)>。发行者是公钥或者是访问控制列表的保留字“Self”，主体已在前文定义，委派比特仅仅是一个布尔值，授权域为应用相关的许可，有效性被定义为截至开始日期和截至结束日期。证书校验链算法使用两个 5 元组作为输入，输出一个 5 元组。特权域和有效期域只能由各自的交叉算法来修改。

```
Input: < Issuer1 , Subject1, D1, Auth1, Val1 >
      < Issuer2 , Subject2, D2, Auth2, Val2 >

IF Subject1 = Issuer2 AND D1 = TRUE
THEN output < Issuer1, Subject2, D2,
               AIntersect(Auth1, Auth2), Vintersect(Val1, Val2) >
```

SPKI 是一个以密钥为中心的 PKI，它是 X.509 的一个有趣的替补规范，至少从证书理论角度看它是如此。SPKI 标准化了某些自动策略决策，比如说授权。因此，我们必须得执行检查以确保这些规则符合给定的组策略。不能够通过直接引用加密的密钥去解释组策略。

15.7 信任管理

一个请求访问资源的主体(同第 4 章的定义)出示一些列的凭证(credential)。凭证就是安全证明。安全策略声明要授予此次访问请求需要哪些凭证。主要考虑 3 种因素：已授予主体的凭证集；为得到访问权限还需要的凭证集；评估需要的凭证是否包含于已授予的凭证的算法。

在基于身份的访问控制中，用户名和密码是典型的凭证。授权的和请求的凭证常在主体访问令牌和对象的访问控制表等明确的地方。检查凭证是否充足的算法相对就很简单：遍历访问控制表，如果所有的凭证都被找到则授权，否则拒绝授权，并且引用监控器在作决定时不请求第三方的帮助。

在开放环境下，要找到授权的以及要求的凭证可能会变得很复杂。因此在做访问决策的时候第三方就有可能加入。考虑如下假设的一个示例：XY 这两个电信提供者达成了服务级别的协议，允许来自 X 的用户访问 Y 提供的服务。Y 无法从 X 获得 X 的客户清单，但是 X 可以把用户的证书和校验密钥交给 Y。来自 X 的客户在请求 Y 的服务时出示他们的证书，Y 回调 X 去执行在线的证书检查。X 的回复将作为 Y 的决定输入。

信任管理是由 Blaze, Feigenbaum and Lacy(1996) 针对一个更通用、更灵活的访问控制方法提出的。在本节的讨论中，我们将重新定义一些术语。在 SPKI 中，主体可以是公钥，我们可以直接授权密钥持有者所能执行的操作，没有必要验证执行访问控制的主体，策略授权的行为用应用特定的行为描述语言格式定义。断言将公钥绑定到行为谓词(predicate)上。断言本质上就是

策略声明,在请求执行这些行为的数字签名可以用断言中给定的公钥验证时,或者在请求的行为满足谓词时,断言就会授权行为。谓词可以使用任何安全的解释语言来写。有了谓词,我们可以说明比访问控制列表更复杂的评估算法。凭证是经数字签名的断言,它们产生授权证书。策略是本地存储的断言,它们是受信的根(授权)。信任关系反映了接受第三方的凭证的策略决策。信任管理为策略,凭证和信任关系提供了一个通用语言。PolicyMaker 是第一个信任管理系统(Blaze, Feigenbaum and Lacy, 1996), KeyNote 是一个用于因特网的信任管理系统(RFC 2704)。

在一个信任管理系统中清楚信任的含义很重要。当我们将信任延迟交付给第三方时,要做一个策略决策来赋予他们做断言的权力。在评估行为请求时我们会考虑这个断言。在实际生活中,信任关系可能是基于不同方的契约关系,也有可能是基于口语语义的信任,但是将进入信任关系(可能与信任无关)的基本原理与信任管理系统的操作方面分开会更加的安全。

访问请求由策略决策点(又名一致性检查器(compliance checker)或者信任管理引擎(trust management engine))评估,其输入为请求 r 和一系列的凭证 C ,并且在回答如下这样的问题时会引用本地策略 P 。(Blaze, Feigenbaum and Lacy, 1996)。

可能的答案有“是”、“否”、“未知”或者一个表明需要更深入检查的标识。在断言语言的昂贵性和符合性检查器的复杂性上,需要一个折衷。依赖于语言的符合性检查器有可能出现无效的情况。我们同样需要考虑用户怎么才能找到适合的凭据使得其出示的证据符合给定的策略。服务端是否需要提供信息,是否需要发布所有的策略?策略本身属于敏感的信息。应当有某种算法可让用户计算他们的请求需要的证据集或者有一种用户可以引用凭证链发现服务(credential chain discovery service)算法。

为了给这个问题一个更大的思考空间,不妨考虑这样的一种联合环境(federated environment):有几个组织合作,但是它们都有各自的安全策略,这样就不再存在只有单独的一个策略设置实体的状况。策略间就必定会发生冲突。在这样的条件下,你就不得不考虑怎么解决冲突,以及不存在冲突又该怎么做。

15.8 数字版权管理

在技术上说,数字版权管理(DRM)指在客户机器上执行售主的策略。这和传统的访问控制范例背道而驰,系统上执行的策略不再由系统所有者制定而是外来者。对手不再是试图颠覆系统的外来者,而是一个试图绕过安全策略的所有者。安全的目标是保证由外来者解释的访问控制系统的完整性。

为了实现这个目标,需要想法让拥有者改变策略设置变得困难。如果对手是一个技术老手,保护机制可能不得不深入到硬件级别。或者在允许下载前,内容提供者要求一份有关目标机的硬件和软件配置的可信报告。这个方案可以在运行证据协议的可信平台模块上实现。(参见 12.6 节)。

从技术角度看,这些挑战非常有趣。一种高级的在内容中嵌入有关内容所有者或者客户的信息的技术出现了,它就是数字水印技术。但是最大的问题就是修改或者移除数字水印的困难,以及对音频视频内容的影响。在分布式的商业模型中,DRM 是否是一个有用的基础的话题已成为了一个热点。

15.9 深层阅读

要知道更多有关 Java 和 .NET 安全的细节信息,可参考 McGraw and FeHen(1997)和 McGraw La Machia et al. (2002)的著作。对历史感兴趣的读者可以查阅 McGraw and Felten(1997)的著作去了解在 Java 安全早期的发现的问题。作为 DRM 历史背景,拷贝代码和拷贝保护之间的竞争趣

事，可以参看 Grover(1992)的著作。当前的有关基于代码的访问控制研究请参见堆栈调查的替换，比如 Fournet and Gordon(2003)的著作，亦可参见策略管理，比如 Besson et al.(2004)的著作。

15.10 练习

练习 15.1 将当前浏览器的安全设置文档化，你的系统上的安全相关信息存放在什么地方？

练习 15.2 试在你的系统中找出 cookie 文件和设置 cookie 存储策略的选项。

练习 15.3 考虑一个使用预保留 cookie 保存会话状态的应用程序。一个攻击者怎么才能劫持会话的其他用户？这个程序怎么才能保证安全？

练习 15.4 分析将访问规则存储在证书而不是访问控制列表中的优点。

练习 15.4 可以使用程序集的 hash 值或者是程序集的数字签名作为其身份，试比较这两种方法各自的优点和缺点。

练习 15.6 考虑一个调用它自己 N 次的递归函数。比较没有尾调用清除和有尾调用清除时堆栈遍历的性能。

练习 15.7 考虑如下的策略：由于职责独立的原因，并不允许所有的实体去执行那些可以授权给其他实体的许可，那么 SPKI 要怎么改进才能支持这样的策略。

练习 15.8 给出一个不是因为信任的原因将访问决策延迟交付给第三方的应用实例。

练习 15.9 在某些环境下要保护运行的系统上的移动代码是可取的。那么这个目标要达到什么样的程度？列出那些可以实现的保护属性和不可能实现的保护属性。

第 16 章 移 动

首先得到广泛的认可的移动信息技术服务是第二代数字手机网络。自此以后，移动服务的数量大大增加，而且使用了不同的基础技术来支持更广泛的应用。移动服务带来了新的安全挑战。其中的有些挑战源于技术。通过无线电传输的信息可能被第三方截取。当一个设备改变它的接入点时，安全会话应该持续。若要访问一个无线网络，你不必将电缆插入插槽，仅需要放入接入点的范围内即可。对楼群和房屋的物理访问控制不再是禁止未授权的用户访问的有效屏障。其他的一些挑战源于应用。通常情况下，对用户提​​供某种服务的实体与管理本地网络服务访问的操作者不同。必须要考虑用户、网络运行商、服务提供商的安全的重要性，这也可能是法律实施代理的要求。

目标

- 研究特定的对移动服务的新的安全挑战和攻击。
- 概述对采用不同移动服务的安全解决方案。
- 给出一些使用加密机制的新方法。
- 讨论在 TCP/IP 网络中的位置管理的安全方面。

16.1 引言

第一代的模拟移动手机系统提供了直接拨号，蜂窝间的自动交递以及呼叫转移功能。在安全内容方面，以下是一个值得注意的有趣的现象，犯罪分子试图用后一个特征（即呼叫转移）来编造托辞，因此电信专家被法庭传讯来解释：为什么回答一个特定电话的人没有必要一定要在电话呼叫的时间、在某个特定的地点。用于认证的简单的挑战-应答协议以明文的方式传输相关机密会导致某些网络遭到费用诈骗。虽然存在语音流量的模糊性，但正如第 1 章所提到的，没有很好的保护措施来防止窃听。

16.2 GSM

为了对抗第一代网络的低安全性，欧洲于 1982 年成立了欧洲邮政电信会议（Conference of European Posts and Telegraph, CEPT）的研究性组织移动通信特别小组（Groupe Spécial Mobile, GSM），来制定第二代移动网络系统。这个数字网络的设计目标是：良好的通话质量、低价格的终端系统、低廉的运行费用、支持国际漫游、支持手持移动设备、与 ISDN 兼容性和支持新的服务（如短消息服务）。在 1989 年，GSM 责任移交到欧洲电信标准化协会（ETSI）接管。1990 年颁布了 GSM 规格 Phase I，GSM 也被重新命名为全球移动通信系统。

如果联系政治环境对 GSM 技术发展的影响，我们可以更好地理解一些 GSM 技术方案的设计决策。一个国际性通用系统必须要同时考虑各个国家的不同法规以及公用系统中不同的加密方法。对强密码学的使用限制使用依然是一个具有广泛争议话题，这个话题持续到二十世纪九十年代中期。而且，法律实施权威要求获得与固定网络中相似的窃听移动网络的权利。

GSM 的主要安全目标是为了防止费用诈骗（对服务的未授权使用）和对语音通信以及无线电信道的信号数据的保护。一旦是在固定的网络上通信，就不需要额外的密码保护。同样，这也是

对物理安全的一点贡献。它使得追踪被盗的终端设备成为可能,而这个特性不经常用到^①。

16.2.1 部件

每个 GSM 的用户都是一个归属网络(home network)的一个订阅者。一个服务被请求的网络被称为被访网络(visited network, 亦称为服务网络。一个移动站(Mobile Station, MS), 即手机由移动设备(Mobile Equipment, ME)和用户识别模块(Subscriber Identity Module, SIM)组成。SIM 是一块智能芯片, 它在 MS 中进行加密操作以及存储相关加密密钥。SIM 也可能包含用户其他相关的个人数据, 像个人电话簿以及提供与移动设备的个人独立性。在网络方面有基站(Base Station, BS), 移动交换中心(Mobile Switching Center, MSC), 用户的归属位置寄存器(Home Location Register, HLR), 认证中心(Authentication Center, AuC)和来访者位置寄存器(Visitor Location Register, VLR)。HLR 和 VLR 管理用户的路由和漫游信息。认证中心管理用户的与安全相关的信息。不同网络运行商之间的关系通过服务水平协议(Service Level Agreement, SLA)来管理。

GSM 用户的标识就是国际移动用户标识(IMSI)。用户和 HLR/AuC 共享一个 128 比特的用户认证密钥 K_i 。SIM 存储 K_i , IMSI, TMSI(后面解释), 以及当前 64 比特的密钥 K_c 。算法 A3 和 A8(后面解释)应用在 SIM 中。PIN(个人识别码)控制对 SIM 的访问。当三次输入错误的 PIN 码时, SIM 就被锁上。可以用个人解密密钥(Personal Unblocking Key, PUK)解锁。

16.2.2 临时移动用户标识

当一个 MS 接入网络时, 他必须通过一些方法来识别自己。如果每次呼叫中使用一个固定的标识, 即使通信被加密, 用户的行动仍有可能被追踪。为了更进一步保护用户隐私, 当且仅当 MS 和 GSM 网络作初始化联接时才传送未被加密的 IMSI。因此, 一个临时移动用户标识(Temporary Mobile Subscriber Identity, TMSI)被分派到被访问的网络以及被用在 MSC 移动交换中心的整个范围。IMSI 不用来标识无线通路。VLR 存在一个从 TMSI 和位置区域标识(Location Area Identity)到 IMSI 的映射 $\{\{TMSI, LAI\}, IMSI\}$ 。当 MS 进入另一个 MSC 时, 就会分配一个新的 TMSI。如果信号程序允许, 传输移动用户身份信息的信号信息元件就会在无线线路上被加密。

经验教训

位置信息的保护是一个安全问题, 特别是对于移动服务来说。固定标识会泄露移动站的运动信息。

16.2.3 密码算法

GSM 使用对称密码学来进行加密和用户认证。曾经考虑过使用公钥加密技术, 但是在 1980 年做出决策的时候, 公钥密码技术还不是一个可行的选择。有 3 种加密算法: 认证算法 A3 和用在信号和用户数据上的加密算法 A5, 以及密钥生成算法 A8。这些算法是不对 GSM 谅解备忘录的成员公开的, 但是最终还是被泄露或逆向工程了。

算法 A3 和 A8 被用户和家庭网络共享。因此, 各个网络都可以选择自己的算法 A3 和 A8。只是它们的输入和输出的格式必须给定。算法 A3 和 A8 从随机口令 RAND 和密钥 K_i , 计算一个鉴权响应符号 RES 和一个密钥 K_c 。处理时间应该在最大值之内, 如算法 A8 在 500 毫秒之内。算法 A3 和 A8 的各种建议由 GSM/MoU 管理, 其中的一个建议为算法 COMP128。这个算法安全

① 在联合王国的犯罪统计中, 学生间的手机盗窃成为一个不可忽视的问题时, 服务提供者被强烈建议实现跟踪被盗手机机制。

性较弱,易于受到从 SIM 卡中窃取密钥 K_i 的攻击。用这个密钥就可以复制设备。慎重的服务提供商还采取了比 A3 和 A8 算法更安全的加密算法。并且易于受到从 SIM 卡中得到的密钥 K_i 的攻击。用这个密钥就可以复制设备。慎重的服务提供者会在 A3 或 A8 中使用更多的安全选择。

所有的用户和网络运行商共享算法 A5。这个算法必须被标准化。该算法有两个版本: A5/1 和安全更低的输出版本 A5/2。尽管这个算法没有官方的详细说明书,但是两个版本的破解算法都已经出版了。

16.2.4 用户身份认证

有两种情况网络将触发用户认证,一是在重启 MSC/VLR 后第一次网络访问时,或者是在用户申请访问服务时,比如一个移动呼叫的建立或者终止的请求。如果用户申请改变移动交换中心和移动位置寄存器中的与用户相关的信息,如关于 VLR 的改变的位置更新或密钥序列号的不匹配那么也会执行用户认证。

图 16-1 描述了认证过程中的信息处理流程。从 ME 到 VLR 的初始信息包含了用户标识符: TMSI 或者 IMSI。VLR 把 TMSI 映射到 IMSI,并将 IMSI 通过固定子网传给 HLR/AuC。AuC 产生一个不可预测的 128 比特的口令 RAND 并计算应答符号 $RES = A3(K_i, RAND)$ 和一个 64 比特的加密密钥 $K_c = A8(K_i, RAND)$ 。三元组 $\{RAND, RES, K_c\}$ 被送到 VLR。VLR 存储了 RAND 和 K_c ,并将质询 RAND 传输给移动站。密钥 K_c 仅在定位区内有效。

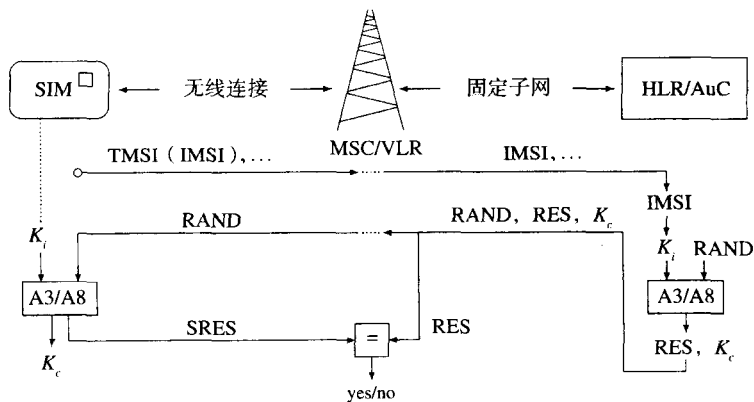


图 16-1 GSM 中的用户身份认证

在 MS 中, 应答 $SRES = A3(K_i, RAND)$ 在用户身份识别卡(“签名”是 GSM 中的术语, 但不是数字签名)中被计算出来并被 MS 传送到 VLR。VLR 比较 SRES 和 RES。如果两个值匹配则认证成功。在被访问网络中, 为了加速随后的认证, 认证中心将几个三元数组 $\{RAND, RES, K_c\}$ 送到 VLR。VLR 则将它们依次用于用户认证。

16.2.5 加密

通常情况下, 所有的语音和非语音通信的无线传输都被加密。基础设施条件决定采用哪种算法, 或者是否不用加密即不提供机密性保护。如果必要, 那么 MS 就会对网络发送信号, 告诉网络自己支持的加密算法。接着服务网络就会基于优先权选择一种它支持的算法, 并将其选择发送给移动站。

密码指示器(ciphering indicator)允许 ME 探知加密未开启并为用户标记它。归属网络操作员可以通过设置 SIM 中相应的管理数据域(EFAD)使该特征无效(GSM 11.11)。如果在 SIM 中没

有禁止它, 则当联接是或者变成不可加密的状况时, 都应该给予用户一个指示 (GSM 02.07, 7.0.0 版本, 1998 发布)。

加密算法 A5 是一个用于 114 比特帧的流加密器。每个帧的密钥都从密钥 K_c 中和通用的 22 比特的帧数据推知。无线连接有相当多的噪音, 因此流加密器比块加密器更加可取。使用块加密器的话, 密码中的一个比特的密码电文的错误都会影响整个明码帧。在流加密器中, 一个比特的密码文的错误仅影响单个明码电文的比特位。据现在的标准, 64 比特的密钥长度是相当短的, 密码分析算法 A5 进一步减少了有效密钥长度。

经验教训

物理网络层的特点与所采用的加密算法是相关的。

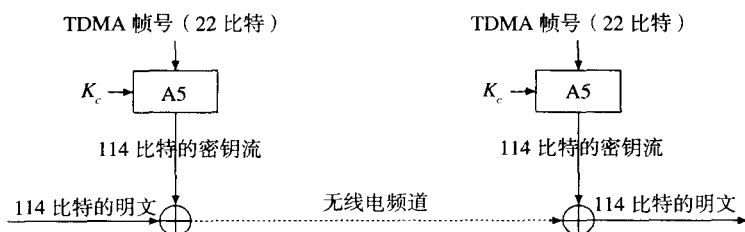


图 16-2 从 MS 到 MSC 的 GSM 帧加密

16.2.6 基于位置的服务

GSM 网络记录了移动设备的位置。这个信息可以用于提供基于位置的服务 (Location-based Service, LBS)。如为监视器提供交通信息。也有一些紧急定位服务, 即给出对某个紧急号码进行呼叫的移动设备的位置。在有的国家这种服务是强制性的, 如美国法律要求移动定位的准确率为 125 米。

16.2.7 小结

在 GSM 中的挑战 - 回应认证不会以明文的方式传输秘密, 因此第一代手机网络的主要漏洞被消除了。通过无线链路的语音通信被加密了, 但是离开基站后, 呼叫以明文方式传播。信号数据的加密是有选择性的, 但是能要求移动设备关闭加密。利用 TMSI 可以在一定程度上保护位置隐私, 但是攻击者可能会使用所谓的 IMSI 捕获器, 让移动设备使用 IMSI 来回复最初的认证。由于网络没有被移动台认证, 所以说这种攻击是有可能的。法律实施代理可以取得用户的移动数据的记录。用户标识 (IMSI) 和设备标识 (IMEI) 是分离的, 这也为追踪被盗设备提供了支持。

人们对 GSM 网络安全性的主要不满在于没有公开一些加密算法以便于对这些算法进行公开研究以及仅提供网络对用户的单边认证而没有用户对被访网络认证。

GSM 安全的总体评价必须考虑技术之外的因素。许多 GSM 诈骗的例子都攻击收入流 (revenue flow) 而不是数据流, 并且不会破坏潜在的技术。在漫游诈骗 (roaming fraud) 中, 签名从归属网络中带出。SIM 卡被运到国外并被用在被访问网络中。诈骗者不为所打电话付费 (软货币诈骗), 但是归属网络必须为诈骗者所使用的服务支付网络访问费用 (硬货币诈骗)。这里存在诈骗者和无耻的网络运行商相互勾结的机会。在费率优惠的诈骗中, 不知情的用户被诱骗去回一个攻击者拥有的费率优惠号码, 这样攻击者就会用现有的收费体系来获得受害者的钱。犯罪分子开通一个费率优惠服务并向他们所拥有的号码做出欺骗性呼叫, 然后向网络运行商领取他们收入的份额, 当网络运行商意识到这个诈骗时, 诈骗者已经消失了。

对策可以被放在用户级上,如用户在回一个电话时要提高警惕性,或者放在法律系统中,如阐明在网络商业模式中可能对用户产生费用的途径。GSM 运营商已经着手使用先进的诈骗探测技术以减少损失,如基于神经网络的早期探测技术。

经验教训

在分析提供给终端用户的安全性时,不要忽略应用程序级的攻击。

16.3 通用移动通信系统

第三代移动通信系统的研发始于 20 世纪 90 年代初。其中的一个系统就是通用移动通信系统 (Universal Mobile Telecommunications System, UMTS)。通用移动通信系统的标准化组织是 3G 合作项目 (3G Partnership Project, 3GPP)。3GPP 的国际合作伙伴有 ARIB (日本)、ATIS (北美)、CCSA (中国)、ETSI (欧洲)、TTA (韩国)、TTC (日本)^①。第一个通用移动通信系统的详细说明在 1999 年出版。通用移动通信系统的安全架构与 GSM 特别相似。每个用户拥有一个作为用户设备一部分的通用用户识别模块,并在归属网络和认证中心共享一个密钥。用户设备要求从被访网络或服务通用分组无线业务节点 (Serving GPRS Support Node, SGSN) 请求服务。

16.3.1 假基站攻击

在 GSM 中,网络没有被移动设备认证。因此,对于使用 IMSI 认证的请求或者是对于关掉加密的请求,移动设备不能区分它们是合法的请求还是来自于一个伪造的基站的请求。为了解决这个问题,一种可能就是要求移动设备和网络的相互认证。但有几种原因阻碍这种方案。移动设备仅在和当地网络联系时有预先的联系,因此对被访网络的直接认证是不可行的。扩展用于相互认证的挑战—回应协议也不能阻止假基站的攻击。攻击者仅需要等待真正的基站的认证完成,接着通过发送比基站更强的信号来和移动设备通信。而且,加密认证机制在噪音信道上的有使用限制。在消息里的任何比特的错误都将引起认证的失败。因此,认证的消息越长,由于传输错误,被拒绝的可能性就越大。

16.3.3 节采用移动通信系统来防御假基站攻击。*Mitchell and Pagliusi (2003)* 给出了一种关于这种机制的更深层次的讨论。

16.3.2 加密算法

各个服务提供商的认证函数 f_1 和 f_2 以及密钥生成函数 f_3 , f_4 和 f_5 是不同的。为 AKA 函数推荐的 MILENAGE 体系,其核心部分包括对 128 位的块的块加密器和一个 128 比特的密钥。无线连接的加密算法和信号数据的完整性检查必须被标准化。通用移动通信系统采用了 KASUMI,拥有带有多个 64 比特的块和 128 比特的密钥的 8 轮 Feistel 密码。KASUMI 被用在用来加密的 OFB 模式的变体,以及用于完整性保护的 CBC-MAC 模式的变体中。所有为通用移动通信系统的算法被公开,并且均已经经过了密码分析。

16.3.3 UMTS 认证和密钥协议

归属网络 (AuC) 和用户 (USIM) 共享一个 128 比特的密钥 K , 并拥有一个同步的序列号 SQN。为了对认证请求进行应答,认证中心产生了随机质询 RAND 和一个期望应答 $XRES = f_2(RAND, K)$ 。认证中心也会计算出一个 128 比特的密钥 $CK = f_3(RAND, K)$, 一个 128 比特的

① 2005 年春季状况。

整数密钥 $IK = f_4(RAND, K)$ ，一个 48 比特的匿名密钥 $AK = f_5(RAND, K)$ ，质询 $RAND$ 的 MAC，以及序列号 SEQ ，和认证管理字段(包含密钥的生命期)(图 16-3)。认证中心接着会创建 $AUTN = \{SQN \oplus AK, AMF, MAC\}$ 并发送认证向量 $\{RAND, AUTN, XRES, CK, IK\}$ 到 VLR/SGSN，它存储 IMSI 的 $\{RAND, AUTN, XRES, CK, IK\}$ 和给用户设备传送 $RAND$ 和 $AUTN$ 。

用户收到 $RAND$ 和 $AUTN$ 后，USIM 首先计算出 $AK = f_5(RAND, K)$ 和 $SQN = (SQN \oplus AK) \oplus AK$ (图 16-4)。接着从 $RAND$ ， SQN 和 AME 计算得到期望的消息认证代码 $XMAC$ ，并与收到的 MAC 值相比较。这样可证实 $RAND$ 和 $AUTN$ 是否是本地认证中心产生的。如果不匹配，USIM 就中断运行的协议，并对 VLR 发出拒绝消息。否则，USIM 就继续协议并检查 SQN 是否能探测重致攻击。如果检查失败了，一个同步错误信号就被送到 VLR 中。因此，是密钥的更新和信号数据的完整性保护的配合而不是通过对服务网络的认证阻止了假基站的攻击。

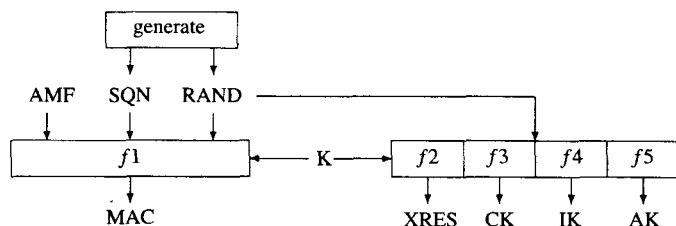


图 16-3 AuC 中认证向量的产生

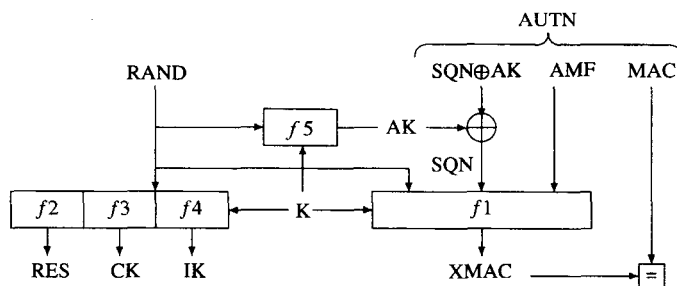


图 16-4 USIM 中的认证

最后，USIM 通过 $RAND$ 和保密密钥 K 计算应答 RES 密钥 K 中的密钥 CK 和 IK ，并访问位置寄存器 VLR 返回应答 RES 。VLR 通过比较 RES 和 $XRES$ 来认证 USIM(图 16-5)。

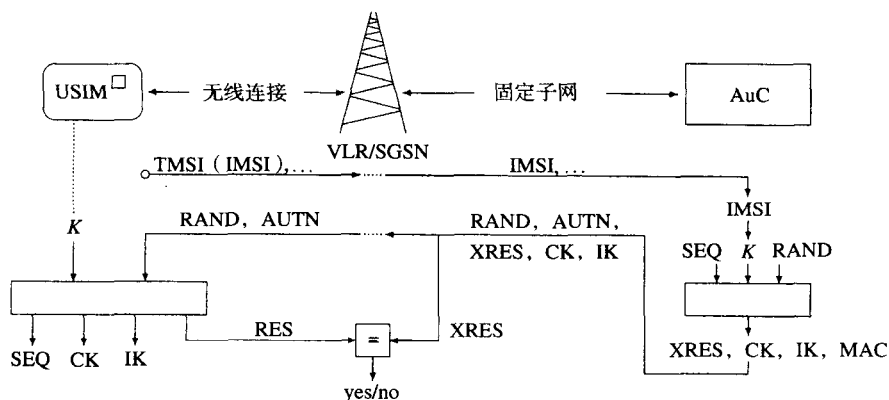


图 16-5 UMTS 中的认证和密钥协议

16.4 移动 IPv6 的安全性

当用户是移动的并且与提供服务的网络运行商没有预先建立联系的时候, GSM 和 UMTS 中的安全问题便是对服务的访问控制。在这一节中我们将用移动的 IPv6 来说明另一种移动问题。当设备改变其位置时, 其他节点怎么确认设备的新地址信息是否正确。在这样的情况下, 标准的补救办法是认证, 但是我们面临两个问题。为了认证消息, 我们必须知道发送实体的标识。在 IP 体系中, 实体是 IP 地址, 我们必须将一个加密密钥和这个地址联系起来。建议用一个 PKI(公开密钥基础设施)来实现这个目的, 但是当前没有我们可以容易使用的 PKI 而且这种情况不会很快地改变。

第二, 数据源认证没有解决我们的问题, 在有线网络的“旧”的背景下, 一个节点可能对其身份说谎。Alice 可能声称自己是 Bob 来得到发给 Bob 的消息。一个移动节点可能谎称自己的身份和位置。Alice 可能声称 Bob 处于她的位置以致打算给 Bob 的通信被传给了她。这仍然是“旧”攻击的变体。Alice 可能声称 Bob 不存在, 以致打算发送给 Bob 的信息丢失了。这两种攻击都可能通过核对 Bob 给出的关于他位置的信息来阻止。

还有一种拒绝服务攻击, Alice 可能声称她在 Bob 的位置, 这样可能给她的信息就送给了 Bob。在轰炸攻击(bombing attack)中, Alice 预定了许多信息, 并将其发送到 Bob 的位置。辨别来自于 Alice 的关于 Alice 的位置信息在这种情况下不会有任何帮助。信息来自于她, 但是她却谎称了她的位置。

轰炸攻击是一个流控制问题。数据被送给没有请求数据的受害者。对原始的位置信息的认证方法不能阻止轰炸。核对接收者收到的数据流是否是想要收到的数据流的方法可能更合适。为了取代原始认证, 我们需要一个一个目的地授权来给它发送信息。流控制问题主要放在传输层处理。但最后还在 IP 层进行处理, 是因为不这样处理的话, 其他的所有传输协议都必须被修改。

经验教训

移动性改变了安全策略的规则。在固定的网络中, 节点可能在不同的会话中采用不同的标识(如: 在 IPv4 中的 NAT()网络地址转换), 但是在每次会话中当前标识都是消息要发送到的位置。对于移动节点, 我们可以把标识和位置看成两个分开的概念。

16.4.1 移动 IPv6

移动 IPv6 详细说明了一个节点和一个位置。一个 128 比特的移动 IPv6 的地址由 64 比特的子网前缀(位置)和一个 64 比特的接口 ID(通过前缀来表明位置范围内的一个标识)。移动节点的 IPv6 地址和固定节点的没有区别。移动节点在归属地址(home address, HoA)中是可以寻址的, 不管它当前连接到归属网络或远离归属网络。归属地址是一个在其归属网络里被分配给移动节点的 IP 地址。移动节点和其本地代理有一个预先配置的 IP 安全联合, 这个安全联合组成一个安全通道。RFC 3776 详细说明用来保护 MIPv6 的 ESP 的使用, MIPv6 在手机和本地代理之间发送信号。当移动节点和一些远离本地网络的外界站点相联系时, 它在转交地址(care-of address, CoA)中也是可寻址的。转交地址是一个带有来自被访问的外界子网前缀的 IP 地址。

16.4.2 安全绑定更新

移动节点的归属地址和转交地址的联合被认为是移动节点的绑定(binding)。离开归属网络后, 移动节点就在归属网络上注册了与路由器的绑定, 要求这个路由器作一个归属代理(home agent)。任何和移动节点相通信的其他节点都被叫做通信节点。移动节点通过绑定更新来通知通

信节点当前的位置信息。通信节点在物理缓存中存储了位置信息。绑定更新更新了绑定缓存中的表项。移动节点和通信节点间的包可以通过归属代理的信道传输,也可以在移动节点的当前位置的通信节点存在绑定时直接传输。

在本质上,绑定更新是一个网络管理任务,但是可以被任何节点执行。如果绑定更新不能被验证,攻击者就可能通过因特网做破坏,包括有线的因特网。毫无疑问,如果任何一个节点都可以参与(涉入)网络管理,这将是一个问题。出于这个安全考虑在一段时间内曾使移动 IP 的因特网工程任务组(IETF)的工作受挫。在安全绑定更新的第一个尝试建议使用 IPsec。这个建议采用了一种古老的办法来解决一个新的问题,并且有严重的缺陷。必须建立安全联盟,但是因特网密钥交换(IKE)协议是一个不适合用在移动的协议。在 Internet 还没有建立全球公开密钥体系(PKI)前,必须建立移动节点和通信节点间的信任链。

因此,为了证实拥有被声明是实体的节点在其声明的位置就需要专用的协议。移动 IPv6 的安全绑定更新协议在 RFC 3775 中有详细说明(图 16-6)。此协议背后的设计论述由 Aura, Roe and Arkko(2002)详细说明。主要的安全目标是:移动性不会削弱 IP 的安全性,对节点的保护不牵涉到交换,如,在固定的因特网中的节点。这个协议也有几个特征来对付拒绝服务的攻击。

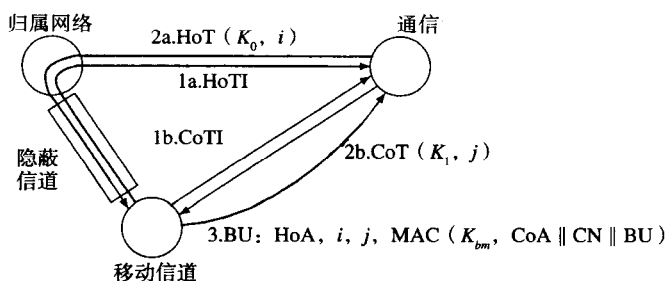


图 16-6 移动 IPv6 的绑定更新

在这个协议中,移动节点首先发送两个初始化消息给通信节点(CN),发送通过本地网络的 Home Test Init(HoTI)消息和直接发给通信节点的 Care-of Test Init(CoTI)消息进行(步骤 1a 和 1b)。通信节点独立回答两个请求。Home Test(HoT)消息包含一个 64 比特的本地注册令牌(local keygen token) K_0 和一个通过移动本地地址传送到移动节点的归属序列号索引(步骤 2a)。包含一个 64 比特的 care-of 注册码以及一个 care-of 序列号索引 j 的 Care-of Test 会被直接发送到声称的当前位置(步骤 2b)。移动节点使用这两个注册码来计算绑定密钥:

$$K_{bm} = \text{SHA1}(\text{本地注册令牌} \parallel \text{可疑注册令牌})$$

以及被 96 比特的 MAC 认证的绑定更新

$$\text{HoA}, i, j, \text{HMAC_SHA1}(K_{bm}, \text{CoA} \parallel \text{CN} \parallel \text{BU})_{96}$$

这个协议不依赖于加密密钥的保密性而依赖于返回的可路由性。通信节点核对是否从被移动节点收到的确认。风险模型认为,密钥 K_0 和 K_1 可以在通道上明文传输。这些密钥可能被解释成一种质询(nonce),这个质询通过把绑定密钥 K_{bm} 把实体与位置绑定起来。在通信安全方面,认证这个术语通常指某类实体标识和通信模型的某个部分之间的链路确认,如,一条消息或一段会话(Gollmann, 2003)。在这种理解方式下,绑定更新协议提供了位置认证(location authentication)。对于窃听通信线路,特别是有线的因特网的攻击者来说,这个协议存在漏洞。如果考虑到有线因特网的安全,可以采用开放式标准框架来保护通信节点和归属代理的通信。

为了有效地对付 DoS 攻击,在协议方面通信应该是无国界的。这就是说,没有必要记住密钥 K_0 和 K_1 。由于对付这个原因,每个通信节点都有一个用来产生送到移动节点的秘密节点密钥

K_{cn} 。这个密钥不必为其他实体共享。每个通信节点在相同的时间间隔都会产生序列号。随机数通过索引来辨识(图 16-6 中的索引 i 和索引 j)。密钥从 MAC 地址的最初 64 位中取得。

$$K_0 := \text{HMAC_SHA1}(K_{cn}, \text{HoA} \parallel \text{nonce}[i] \parallel 0)_{64}$$

$$K_1 := \text{HMAC_SHA1}(K_{cn}, \text{CoA} \parallel \text{nonce}[j] \parallel 1)_{64}$$

通信节点在答复后即可丢弃密钥 K_0 和 K_1 , 因为他在收到最后的确认之后能重新生成密钥。因此通信节点保持的这个状态独立于他收到的 BU 请求的数目。平衡信息流是对抗拒绝服务攻击的另一个规定。可以使用采用一条以上的消息来回应收到的一条消息的协议来增强拒绝服务攻击。因此, BU 的请求被分为两部分。移动节点可以通过一条消息同时传送归属地址和转交地址, 但是接着通信实体可能用两个 BU 的认证来回应一个 BU 请求。

经验教训

在通信安全中, 一般认为被动的窃听攻击比主动的攻击更容易。在移动系统中, 则相反。为了能够在一个特定的移动节点上窃听通信, 他必须在附近窃听。试图干扰本地的操作则可以在任何地方发起。

16.4.3 地址所有权

动态分配地址的方案应该确认一个新的地址是空闲的。这可以通过广播一个消息来询问是否这个网络上有节点使用了这个地址。在蹲守攻击(squatting attack)中, 攻击者谎称他拥有应该被分配的地址, 因此就阻止了受害者从网络中取得分配地址。

我们将描述一个方案, 即一个节点能证明自己“拥有”一个 IP 地址的方案。这个方案有一个不依赖于任何第三方的鲜明的特点, 不管它是归属代理还是证书权威。核心思想是地址所有者创造了一个公钥/私钥对, 并用公钥的哈希值作为一个 IPv6 地址的接口。

为了证明对一个地址的所有权, 声明应该用密钥进行加密。并将其与公钥一同发送给通信节点。通信节点将核对声明上的签名并核对公钥和 IP 地址的联系。地址是用公钥“验证”了的。公钥加密用于未使用公钥基础设施(PKI)。

密码生成地址(Cryptographically Generated Address, CGA)在 RFC 3972 中已详细说明, Aura (2003)将这个思想用于 IPv6 地址。在这种情况下, 哈希值的长度可能被限制在 62 比特, 因为接口 ID 中有 2 比特保留位。为了对一个给定的地址伪造申明, 攻击者不必找到原来的密钥对, 而仅需要一个冲突, 即含有与接口 ID 相冲突的公钥的密钥对。找到与 62 比特哈希值的冲突的非常容易, 因此并不应满足于此, 而是还要找出一种扩大哈希冲突值的方法。一个密码生成地址因此有了一个安全参数 Sec (3 比特的无符号整数), 该字段被编码在接口 ID 的最左边的 3 个比特位。安全参数以 16 比特的增量来增长哈希值。公钥的哈希值 Hash1 和 Hash2 可如下计算:

$$\text{Hash1} = \text{SHA-1}(\text{修改器} \parallel \text{子网前缀} \parallel \text{冲突计算} \parallel \text{公钥})$$

$$\text{Hash2} = \text{SHA-1}(\text{修改器} \parallel 0_{64} \parallel 0_8 \parallel \text{公钥})$$

Hash2 的 $16 \times Sec$ 的最左边的比特必须为零。Hash1 最左边的 64 比特成为了接口 ID, 除了 5 位固定比特外, 仅 59 比特可用。抵制冲突攻击与查找一个 $59 + 16 \times Sec$ -比特的冲突的工作量相当。为了得到一个符合要求格式的 Hash2 的值, 地址所有者必须通过更改修改器做强制的搜寻, 而这是一个随机的 128 比特的数。这个寻找的工作量等同于找到一个有 $16 \times Sec$ -比特的相当与一个确定值(全为零)相等的哈希值。冲突计数初始化为零, 如果在地址空间发生冲突, 冲突计数就会增加。当三次失败后就会发送一个错误报告。

通信节点的工作量是恒定的。为了验证地址和公钥之间的对应关系, 就必须计算两个哈希函数。为了验证一个签名, 就必须验证一个签名, 当然, 在拒绝服务攻击中这可能又是一个难

点, 因为签名认证是一个代价高昂的操作。密码生成地址不能阻止攻击者根据自己的公钥和任何子网前缀中伪造地址。因此, 密码生成地址不能阻止轰炸攻击。为了防御这种攻击, 就要运行一个可返回路由测试来核实接收者是否愿意接受传输。

16.5 无线局域网

无线局域网(WLAN)是用于无线的局域网的一项技术, 该技术的详细说明参见 IEEE 802.11 标准系列。在这个应用中的安全质询不是移动的主要顺序而是无线网络访问的主要顺序。又, 在空中传播的数据的完整性和机密性应该得到保障。应该选择合适的密码算法并定义密钥管理过程。而且, 必须控制对归属网络和移动设备的访问。一个未受保护的无线广域网可能给攻击者使用由网络所有者付费的带宽服务以可乘之机, 或把受害者的系统作为一个攻击第三方的中间站。

无线局域网可以在基础设施模式和自组织模式中运行。在基础设施模式中, 移动终端通过接入点与归属网络连接。在自组织模式中, 移动终端直接通信。本节的安全讨论将集中关注基础设施模式和接入点的访问控制选择。每个接入点都有一个服务集标识(SSID)。一个开放的无线局域网对谁可以与接入点进行连接没有限制的。一个开放的无线局域网不必是无保护的, 应该在其他协议层提供安全机制。公共接入点被作为热点。

为了控制对无线局域网的访问, 接入点应配置为: 不能广播它们的服务集标识, 而要求客户知道它们要连接的访问节点的服务集标识符。服务集标识是取得访问的必须的秘密。这种方法运转得不是很好。服务集标识包含在许多发送信息中, 这些信息可能被攻击者破译, 所以它不该当作秘密。此外, 对不同的访问节点通过被运行商设置的缺省的服务集标识符来进行区分。攻击者常会抱着这些缺省的值一直没有被改变的心态而进行尝试接入, 而且这样做常常是有效的。另外, 更有可能去通过对访问节点的配置, 只允许那些 MAC 地址为某些规定值的移动终端接入网络。这种基于地址的认证方式并不是非常安全。攻击者可能通过监听合法设备的连接来得到合法的 MAC 地址, 接着就用一个伪造的 MAC 地址进行连接。

总的说来, 把访问控制建立在网络连接管理所需要的信息基础上是有问题的, 如, SSID 和 MAC 地址。通常情况下, 当要建立一个连接时, 这些信息就必须在安全机制被起用之前被传送。所以, 让客户发起一个对访问节点的连接并在授权对保护服务进行访问之前对客户进行认证是一个更可靠的策略。图 16-7 所示的通用访问机制表明了这个方法。客户被默认是安装了 web 浏览器。连接到访问节点的客户可以从 DHCP 服务器处得到一个动态的 IP 地址。当客户的 web 浏览器启动时, 初次的 DNS 或 HTTP 请求被中断, 通过 HTTP 会话将客户转入请求用户名和密码的页面。访问节点控制器中 web 服务器将输入的用户名和密码提交到远程用户拨号认证系统(RADIUS)进行认证。一旦客户端被认证, 访问点就可以对客户端的请求应用熟悉的基于身份的访问控制。对客户端和访问点的后继通信的保护是一个独立的问题。对这个方案提供的安全性的分析留作练习。

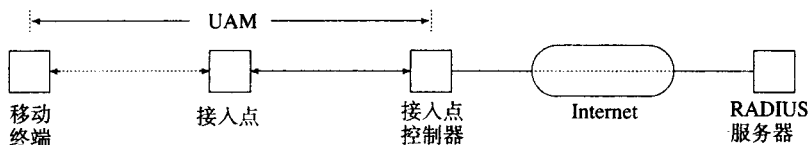


图 16-7 使用 UMA 和 RADIUS 服务器的热点访问

16.5.1 无线对等加密

在 IEEE 802.11 中详细说明了无线对等加密(Wireless Equivalent Privacy, WEP)协议被建议

用来保护在移动终端和接入点之间传输数据的机密性和完整性,并用来认证网络的移动终端。认证基于一个共享秘密。所谓的预先共享式秘密被手动安装在将要访问的所有设备上,和网络中的所有接入点上。这种方法适合于像归属网络这样小的安装上。大多数的局域网的所有终端都使用相同的密钥。

无线对等加密采用流密钥来加密。在流密钥中,密钥不能被重用,因为被相同的密钥加密的明文的异或与两个明文的异或值是相同的。两个明文的异或值的统计特性可以被用在重建两个明文的加密分析,因此也是密钥流。因此,一个 24 比特的初始化数组 (Initialization Vector, IV) 用来使加密随机化。发送者和接收者共享一个 40 比特或 104 比特的密钥 K 。为了传输一个消息 m , 发送者计算出一个 32 比特的校验和 $\text{CRC}-32(m)$, 用 24 比特的初始化向量和产生一个使用流密钥的 RC4 的 64 比特的密钥 $K' = \text{IV} \parallel K$ 。密文。加密文本 c 是 $(m \parallel \text{CRC}-32(m))$ 和密钥流的位异或

$$c = (m \parallel \text{CRC}-32(m)) \oplus \text{RC4}(K')$$

密文和初始化向量被传输给接收者,接收者计算出 $C \oplus \text{RC4}(K') = (m \parallel \text{CRC}-32(m))$ 和核对校验和。为了认证一个客户,接入点以明文的方式将一个 1024 比特的质询发给客户。客户用上面的算法和预先共享密钥来加密口令,并且接入点认证这个响应。

在对等加密中使用的加密机制有两个主要的设计缺陷。首先, $\text{CRC}-32$ 是循环冗余检查,即线性函数被用来检测随机错误但是不能防御目标的修改。当用在一个流密钥的线性加密算法 (异或) 的连接,它不提供任何真正的完整性保护。仅拥有无线对等加密密文但是既没有密钥 K' 或明文 m 的攻击者能够像如下修改明文。令 Δ 是明文的想要的改动。攻击者计算 $\delta = \text{CRC}-32(\Delta)$ 并将 $(\Delta \parallel \delta)$ 加入密文,可参照下式得到一个明文的有效加密 $m \oplus \Delta$:

$$\begin{aligned} (m \parallel \text{CRC}-32(m)) \oplus \text{RC4}(K') \oplus (\Delta \parallel \delta) &= (m \oplus \Delta \parallel \text{CRC}-32(m) \oplus \delta) \oplus \text{RC4}(K') \\ &= (m \oplus \Delta \parallel \text{CRC}-32(m \oplus \Delta)) \oplus \text{RC4}(K') \end{aligned}$$

第二个问题是初始化矢量的大小。只要密钥 K 保持无变化状态,初始化数组便是密钥 K' 的唯一可变部分仅为初始化矢量 IV。攻击者就可以观察一段通信量,直到初始化数组重复,然后尝试重新生成流密钥并建立初始化数组表和通信密钥流。

掌握这些设计缺陷后,在 RC4 的密码分析上便有了些进步。*Fluhrer, Mantin and Shamin (2001)* 描述过寻找密钥 K 的攻击,并且这种攻击从那以后都已经改进了。这些攻击通过一位一位地取得密钥,因此一个 104 比特的密钥并非比 40 比特的密钥更难破解。有的攻击会通过重放加密控制信息来产生更多的流量,以致他们可以取得用来分析密码的足够数据。

16.5.2 WPA

无线对等加密在满足它的安全目标方面做得相当失败。WiFi 保护接入 (WPA) 被用作一个快速的预处理办法。这个方法可用来消除无线对等方案的主要缺点,而不需要对无线局域网的安全体系的重设计。无线应用协议要求在已有无线广域网的硬件上运行。其中也有对访问网络的客户的认证以及动态创建临时密钥进行改进的程序。扩展认证协议 (Extensible Authentication Protocol, EAP) 是一个在 RFC 2284 中详细说明了元协议,支持多重认证方式的 IEEE 802.1X,如令牌卡片, Kerberos, 一次性口令, 证明和公钥认证。

为了完整性保护, $\text{CRC}-32$ 被叫做 Michael 的信息完整性检测码 (message integrity code, MIC) 所代替。初始化向量的长度是原来的两倍,达到 48 比特。临时密钥完整性协议 (Temporal Key Integrity Protocol, TKIP) 创建了密钥体系系列。客户端和访问点需要长的成对主密钥。新的成对临时密钥 (Pairwise Transient Key, PTK) 来源于主密钥,并被用来保护移动终端和接入点的

通信。当 WPA 是使用预先共享主密钥配置 (WPA-PSK) 时, PMK 就可以用密钥生成函数 PBKDF2 (RFC 2898) 进行如下计算:

$$\text{PMK} = \text{PBKDF2}(\text{passphrase}, \text{SSID}, \text{SSID length}, 4096, 256)$$

输入包括一个密码 (推荐用 20 个字符)、接入点的服务集标志和服务集标识的长度。这个输入哈希 4096 次后就返回一个 256 比特的密钥。计算 PTK 的算法采用成对主密钥 (PMK), 两个设备的 MAC 地址, 以及输入时两个设备产生的 nonce。nonce 以明文方式传输。

WPA-PSK 对于猜密码的攻击来说是脆弱的。当受害者和无线广域网联系的时候, 攻击者会记录下交换的信息。接着, 攻击者就猜密码, 并为猜出的密码, 已知的 SSID 的值和 SSID 的长度, 从 PTK' 到 PMK' 的临时密钥, 被截获的 MAC 地址, 和随机数计算一个主密钥 PMK'。已经被记录的加密消息用候选密钥 PTK' 解密。如果结果是有意义的明文, 那猜出的密码就很有可能是正确的。

由于这些设计上的缺陷, WPA 的安全机制就可能没有预想的加密观点那样强大, 但是 WPA 在 WEP 方向上仍然有一定的改进。

16.5.3 IEEE 802.11i - WPA2

在 2004 年, IEEE 802.11i 系列标准详细说明了无线局域网安全机制的全新设计。这个标准就是著名的 WPA2。在 CCMB 模型中, 流加密器 RC4 已经被 AES 所取代, 如反击模型中的 CBC-MAC 模式的计数器模式。WPA2 要求新的硬件。

16.6 蓝牙

蓝牙是无线自组织网络的一种技术, 最初是被用在个人设备如 PC、键盘、鼠标、打印机、耳机和其他外围设备的短距离的通信 (最高可达 10 米), 形成了个人局域网 (Personal Area Network)。该本地自组织网络不要求复杂的密钥管理基础设施。两个设备之间的安全联系通过配对手工确立, 如用户在两个设备上键入一个公用 PIN 码。

为了对两个设备之间的通信进行加密保护, 可由 PIN 码生成 128 比特的连接密钥。连接密钥被用在两个设备之间的后期的认证中。认证由质询-应答协议执行, 与 GSM 认证相似。用来加密消息的临时通信密钥由共享的 PIN 码、PIN 码的长度, 以及发送者产生的随机值和接受者的地址生成。设备认证是访问控制的基础。访问控制客体是在设备上提供的服务 (拨号、文件传输等)。

蓝牙 (Bluetooth) 安全体系是为个人局域网设计的。由于蓝牙的应用和蓝牙技术本身在不断变化, 扩大了通信的范围, 因此还需要研究新的安全体系。针对蓝牙的应用, 我们必须考虑那些利用设备的软件配置所暴露的弱点而进行的应用级攻击。像 Bluesnarf 这种攻击可以从那些在访问控制的实现上存在缺陷的设备上获取个人数据。当蓝牙设备被用来广播某些被请求的标识时, 就需要建立用户的漫游控制。

16.7 深层阅读

本章主要描述了移动和无线技术, 主要集中在各种大家感兴趣的安全问题。本章没有介绍的与实际相关的有趣的安全问题多与个人技术特定方面相关。Niemi and Nyberg (2003) 给出了关于 GSM 和 UMTS 的详细描述。GSM 标准由 ETSI 管理 (<http://www.etsi.org>), UMTS 规范由 3GPP 管理 (<http://www.3gpp.org>)。TCP/IP 网络中的移动问题将在各种 IETF 工作组中讨论。这些讨论的当前状况可以在各自的 IETF 的站点找到 (<http://www.ietf.org>)。无线广域网安全的详细说明参见 IEEE 802.11。蓝牙成员的官方站点是 (<http://www.bluetooth.org>)。

16.8 练习

练习 16.1 电话服务提供商必须要处理客户的抱怨, 这些客户称没有打过一些似乎从他们的手机打出的兑奖电话。可以提出什么样的措施来解决这样的问题呢。

练习 16.2 试讨论一下对具有国际漫游移动系统中的窃听被批准后所引发的问题。

练习 16.3 假如一个 128 比特的 MAC 地址被用在信道错误率为 1:1000 的消息认证。当消息长度分别为 1K 比特, 2K 比特和 3K 比特时, 由于传输错误而否认一个消息的可能性是多少? 当比特的错误率达到 1:100 的时候可能性又是多少? 在这两种错误率下, 分别使用 32 比特的 MAC 时, 可能性又是多少?

练习 16.4 在传输层已经建立了一个会话的节点在会话期间改变了他们的 IP 地址。在这样一种情况下, 会话怎样在传输层被保护。对这个解决方案中提到的那些具有普遍性的安全问题进行分析, 并对 IETF 中当前正在被讨论的建议进行分析。

练习 16.5 设哈希值的计算时间为 1 毫秒, 相对于攻击者在值 $Sec = 1, 2, 3$ 时所付出的努力而言, 地址所有者创建一个有效的 CGA 需要付出怎样的代价?

练习 16.6 试考虑有这样一个繁忙的接入节点, 该节点以 11Mbit/s 的 IEEE 802.11b 的数据率, 发送 1500 字节的包。攻击者必须要花多长时间等待初始化向量开始重复? 如果初始化向量是随机产生的, 那么攻击者为第一次冲突等待的平均时间是多长(两个包用相同的初始化向量加密)?

练习 16.7 WEP 的质询-应答协议用明文发送质疑并加密应答。如果质疑被加密应答以明文传送, 会有什么不同?

练习 16.8 试描述一个这样的攻击, 在这个攻击中, WEP 加密的 IP 包还需要被发送到攻击者指定的目的地。

练习 16.9 给出一个被 UAM 和采用 EAP(RFC 2284)的 802.1X 提供的安全服务之间的比较。

第 17 章 数据库安全

一个数据库不只是用来存储数据，它还为用户提供信息。因此数据库安全就不应只关注如何保护敏感的数据，它也应研究一些机制，让使用者在可控的范围内检索资料。在这里强调了两个把数据库的安全和操作系统的安全区分开来的主题，我们对信息的控制访问应该多于对数据的控制访问。尽管对数据的保护仍然是一个重要的问题，但是我们一定要把重点放在控制主体请求访问上。

目标

- 分析数据库系统所特有的安全问题。
- 理解怎样把各种观点用在关系数据库的访问控制之中。
- 分析在统计数据库中保护信息安全的问题。
- 检查在数据库管理系统中和在隐藏着的操作系统中安全机制之间潜在相互作用。

17.1 引言

数据库是数据以某种有意义的方式排列的集合。数据库管理系统(DBMS)把数据组织起来并且给用户提供检索信息的方式。如果对信息的访问控制完全失控的话，那么用户就很可能不会(可能是被迫)把某种数据放进数据库中。从而数据库提供的有用的服务就会变少。比如：数据库常常会存储一些私人的信息，如公司中员工的记录，学校中学生的记录，以及税务部门的税收记录。因此，从很早以前起，数据库的安全就在计算机的安全中占据相当重要的作用。数据库的安全有着特殊的地位，因为它不同于操作系统的安全，下面将给出这些说法的具体说明。

操作系统管理数据。用户可以调用操作系统的函数来创建文件，删除文件，或是打开文件进行读和写的访问。这些操作中没有一个考虑到了文件的内容。更恰当地说，类似的访问控制的决定是由操作系统决定的。这些决定依赖于对用户的识别，文件属性的定义，访问控制列表，安全标签等，但是并不依赖于文件的内容。这样做的起因并不是在于一些基本的安全理论，而仅仅是一种合理的工程考虑。

数据库中的表项携带有信息，数据库的用户执行的是与数据库中表项内容相关的操作。数据库中最典型的作用就是数据库的检索功能。因此，数据管理系统在考虑到数据库表项的具体内容后做出访问控制决定。这样做比较合理。一个很常用的例子就是工资数据库。在该数据库中，达到一定上限的工资是要保密的。总而言之，在人-机标尺之中，数据库的安全是更偏重于用户这端(见图 17-1)。

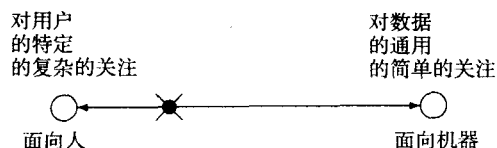


图 17-1 数据库的安全在人-机标尺上的位置

粗看起来,保护数据库中的敏感信息似乎很容易,在工资数据库中,你可能会在查询语句中添加一些检索工资额的条例。如果你知道要保护的是哪些数据,这种方法当然是可行的。但是,入侵者可能会对许多不同的信息片断感兴趣。下面是一些可能的信息源。

- 精确的数据:存储在数据库中的数值。
- 边界:数值(如工资)的下界和上界可能已经是有用的信息。
- 消极结果:例如,如果一个数据库包含了许多刑事犯罪的信息,那么这种涉及某一特定人物犯罪的信息就是很敏感的信息。
- 存在性:数据的存在性本身就可能是敏感的信息。
- 可能值:能够从其他的查询结果中猜测出某种信息。

最后,你不得不要防止一些意外事件发生。如果数据库允许统计查询的话,那么保护信息就会变得困难起来。例如,一次统计查询就会返回所有工资的总额,或是所有工资的平均值。像这种查询的巧妙结合反而可能会暴露你想保护的信息。这个话题可将在 17.4 节里面深入讨论。

你可能被警告过许多次了:有许多途径可以使敏感信息从你的数据库中泄漏出去。当然,你应该对安全性有足够的重视。而且也不要忽视了你的数据库要为某种有用的目的提供服务。如果你采取太严格的政策,即使敏感的信息不会被暴露,而不提供对数据的访问,那么你会减少数据库的价值。你因此不得不尽量做到精确,在暴露那些不重要得信息的同时,保护好那些敏感的信息。

数据库的表项携带了实体外部到计算机系统的信息。如,在库房中的库存,学生的考试结果,银行账户的收支,和飞机舱里面可用的座位数。数据库的表项应该非常准确地反映客观事件。数据库的安全结合了特定应用程序的整体性保护来达到。

- 内部一致性(internal consistency):数据库中的表项应该遵循一些事先约定好的规则。例如:库存量不能低于零。
- 外部一致性(external consistency):数据库中的表项应该完全正确。比如,在数据库中的库存量应该与实际的库存量保持一致。当你在更新数据库的数据的时候,数据库管理系统能够帮助你避免一些错误,但是你也不能完全指望单靠它来保持数据库的一致性状态(consistent state)。这种属性也称为精确性(accuracy)。

在第 2.4 节中的分层模型中,数据库管理系统被放在了操作系统顶部服务层上。数据库管理系统必须满足特定的数据库安全需求,这种需求是操作系统所不能应付处理的。数据库管理系统可以结合操作系统之内的一些机制来强制保证安全性,而如果在操作系统之内没有恰当的控制方法,或是自己牵涉进入操作系统后变得很麻烦的时候,数据库管理系统就会自己独立地强行保持安全性。而且,数据库管理系统可以在应用层中,作为一个定义安全控制的工具。图 17-2 说明了这样的事实:数据库的安全包括了各个不同的抽象层的安全机制。

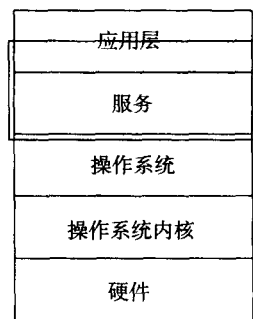


图 17-2 数据库安全的位置

17.2 关系数据库

在构成数据库的各种模型中,当今使用最为广泛的是关系数据库。再次地,我们认为读者对关系数据库中基本的概念已经比较熟悉了,所以就只给出关系数据库的简单介绍。详细的介绍可以参考 Date(1990)。

关系数据库 用户会把关系数据库看成是各种表(只有表)集合的一种数据库。

这种定义指的是用户对关系数据库的一种感知，而不是数据库实际的物理组织方式。这也恰好成为我们讨论数据库安全恰当的抽象层基础。

形式上，关系 R 是 $D_1 \times \cdots \times D_n$ 的子集。而其中 D_1, \cdots, D_n 是 n 个属性中的域，关系中的元素是 n 元组 (v_1, \cdots, v_n) ，其中 $v_i \in D_i$ ，比如第 i 个属性就是来自元素 D_i 。元组中的元素通常称为字段。当一个字段未包含任何值的时候，就用一个特殊的空值来表示。空值的意义是没有任何输入，而不是“该项为未知”。

在图 17-3 中的关系可能是一个旅行社数据库的一部分。关系 **Diary** 有四个属性，分别是 **name**，**day**，**flight**，**status**，各个域的范围如下。

- **name**: 所有有效用户名。
- **day**: 一周中所有天数，如: Mon, Tue, Wed, Thu, Fri, Sat, Sun。
- **flight**: 航班号，两个字符后面加上 1~4 个数字。
- **status**: 公务 (business) 和私人 (private)。

Name	Day	Flight	Status	Flight	Destination	Departs	Days
Alice	Mon	GR123	private	GR123	THU	7:55	1-4-
Bob	Mon	YL011	business	YL011	ATL	8:10	12345-7
Bob	Wed	BX201		BX201	SLA	9:20	1-3-5-
Carol	Tue	BX201	business	FL9700	SLA	14:00	-2-4-6-
Alice	Thu	FL9700	business	GR127	THU	14:55	-2-5-

图 17-3 Diary 和 Flight 的关系

在关系数据库中用来描述信息查询和更新方式的标准语言称为结构化查询语言 (SQL) (International Organisation for Standardization, 1992)，SQL 语言包括以下一些操作。

- **SELECT**: 从一个关系中查询数据，

```
SELECT Name, Status
FROM Diary
WHERE Day = 'Mon'
```

返回的结果:

Name	Status
Alice	private
Bob	business

- **UPDATE**: 更新关系中的字段，

```
UPDATE Diary
SET Status = private
WHERE Day = 'Sun'
```

把星期日的所有旅行团标记为私人旅行团。

- **DELETE**: 从一个关系中删除元组

```
DELETE FROM Diary
WHERE Name = 'Alice'
```

从 **Diary** 中删除 Alice 所有的旅行。

- INSERT: 将元组插入到一个关系中,

```
INSERT INTO Flights(Flight, Destination, Days)
VALUES('GR005', 'GOH', '12-45-')
```

在关系 Flights 中插入一个元组, 而字段 Departs 域仍然没有给出。

在所有的情况下, 都有可能出现更复杂的情况。编写这本书的目的并不是用来解释 SQL 的复杂性的, 在此, 我们仅给出一个例子予以证明其复杂性。要找到谁要去 Thule, 就要执行下列语句:

```
SELECT Name
FROM Diary
WHERE Flight IN
( SELECT Flight
FROM Flights
WHERE Destination = 'THU')
```

关系常常用表来表示。属性对应的是表中的列, 用的是属性的名字作为列的表头。在表中的一行对应的是数据库中的一个元组(记录)。在关系模型中, 关系并没又包含指向其他表的连接和指针。表(关系)间的关系只能用另一关系来表示。在关系数据库中可能存在着不同种类的关系, 如下。

- 基本关系: 也称为实际关系(real relation), 它们被命名为自主的关系。它们独立存在, 而不是派生于其他关系, 并且它们拥有自己存储的数据。
- 视图: 被称为导出关系。它们由其他已命名的关系定义; 它们自身并不存储数据。
- 快照: 像视图一样, 它们也被称为导出关系, 也是根据已经命名的好的其他关系定义出来的, 但是它们拥有自己独立的存储的数据。
- 查询结果: 一个查询的结果, 它们可能有也可能没有名字, 它们并不是常驻留在数据库中的。

例如: 在表 Diary 中查询谁以及何时旅行的快照查询操作如下:

```
CREATE SNAPSHOT Travellers
AS SELECT name, day
FROM Diary
```

17.2.1 数据库的关键字

在每一个关系中, 我们必须用一种特殊的方式识别出所有的元组。有时, 单独的属性就能够作为这样的标识。也有可能满足这种目的的多个属性中作选择。而另一方面, 也有可能要用多个属性组合起来才能构造出唯一标识来。

主键是指一个关系中唯一的并且是最小的标识。关系 R 中的主键必须满足下面的条件。

- **唯一性:** 在任何时候, 关系 R 的两个元组都没有相同的 K 值。
- **最小性:** 如果 K 是组合型的, 那么要满足唯一性的前提, K 中的每个成员都不能少。

在关系 Diary 中, name 和 day 的组合就可以看成是主关键字(假定对顾客而言, 每天只能跟一个旅行舍出团)。在关系 Flight 中, 主键是 flight number。

每个关系必须有一个主键。因为没有一个关系会包含两个完全相同的元组。这是直接来自于我们对关系的定义的。当一个关系中的主关键字在另一个关系中作为属性出现时, 那么该属性就称为外键。在这个例子中, 关系 Flight 中的主关键字 flight number 就是关系 Diary 的一个外键。

17.2.2 完整性规则

在关系数据库中,可以通过定义完整性规则来强制保证内部一致性,而且可以帮助维护外部一致性(精确性)。这些规则中的绝大部分都是只针对应用程序而言的,但是也有两条规则是关系数据库所固有的。

实体完整性规则 基本关系的主键值中,不允许主键的一个属性或是主键的一个子集是空值。

这条规则允许我们找到基本关系中的所有元组。在基本关系中的这些元组与现实中的实体是相互对应的,并且我们不会在一个数据库中建立一个不能识别出来的元组。

引用完整性规则 数据库中不得包括不匹配的外键字值。

一个外键值声明了一个对其他某个表的表项的引用(reference),一个不匹配的外键值是一个在引用表中不作为主键出现的值。它是一个对不存在的元组的引用。

除了这两条规则外,我们还需要更多的特定应用的完整性规则。这些完整性规则很重要,因为它们能保证数据库处于可用的状态下。很典型地,你会用这些完整性规则来做下面的事情。

- **字段检查:**防止错误的数据输入。在我们的例子中,我们通过一条规则,检查输入的值是 business 还是 private 来防止用户随意将值输入到 Diary 关系的 status 属性中。
- **范围检查:**在统计数据库中,可能需要一些规则,来检查查询的结果是否由足够的例子计算出来。请先看看图 17-4 中的 Students 关系。你可能会定义一条规则,在样本不超过三个的情况下,就返回平均成绩的默认值 67。
- **一致性检查:**不同关系中的表项可能指的是外部世界中的相同的方面,因此也应该在这个方面表示出一致性。在我们的例子中,我们能够检查出旅客出行的日期是否与它们的航班起飞的日期保持一致。Alice 的乘坐的是周一的 GR123 航班,就应该与该航班在周一和周四起飞保持一致。而 Carl 预定的是周二的 BX201 航班,就与该航班在周一,周三,周五起飞的事实不相一致。一条像这样的比较各自的域的完整性规则,就可以防止旅行社发生这样的错误。

像这样的完整性规则由应用层控制。数据库管理系统为指定和强化这些规则提供了基础。比如,完整性触发器(integrity trigger)就是一个用来附带在数据库中的某个对象上,检查该对象的完整性属性的程序。当有一个更新(UPDATE)、插入(INSERT)或是删除(DELETE)操作试图改变该对象的时候,这个程序就会被触发,执行相应的检测。

在指出了机密性和完整性之间潜在的冲突性之后,我们将不对该话题进行更深入的探讨。当评估一个完整性规则而要访问敏感的信息的时候,你可能面临一个两难的抉择:是保护敏感的信息而不完整(不正确)地评估规则,还是为了维护数据库一致性而泄漏一些敏感信息。

17.3 访问控制

要保护敏感信息,数据库管理系统就不得不控制用户如何访问数据库。要弄清楚这些控制怎样实现的,你就应该考虑分两个层次来访问数据库,这两个层次分别是:

- 基于基本关系上的数据处理操作。
- 复合操作,如:视图或快照操作。

返回到 2.4.1 节,你就可以从两个角度使用如下规则来看待访问控制:

- 限制用户可用的操作;

- 为每个独立的数据项定义保护需求。

在数据库管理系统中,对复合操作的控制管理着用户如何使用数据库。另一方面,基本关系的操作检查保护数据库中的表项。通过决定你想要的控制的访问操作类型,你也影响了将要被执行的策略的重点所在。相反地,策略的重点,也暗示出要控制的操作类型。不管你作出何种选择,有两种特性是要力争达到的。

- **完整性:** 保护数据库中的所有字段。
- **一致性:** 在管理访问数据项时,不能存在相互冲突的一套规则。

数据库中不应该存在因为访问方式不同而导致有不同的访问控制的元素,这样的策略才是安全的策略。不应该避免合法的访问请求,同时也不应该有可以绕过特定访问策略的方法。

17.3.1 SQL 安全模型

基本的 SQL 安全模型应该遵循相似的模式。它根据三个实体来实现自由访问控制。

- **用户实体:** 数据库的用户。在登录进程中,用户身份要得到认证,数据库管理系统执行它自己的登录进程,或是接受操作系统认证的身份认证。
- **行为实体:** 包括选择(SELECT),更新(UPDATE),删除(DELETE)和插入(INSERT)。
- **对象实体:** 表,视图以及表和视图的列(属性);SQL3 进一步把用户定义的数据类型也包括进来了。

用户调用了对象上的操作,并且数据库管理系统不得不决定是否允许请求的操作。当用户创建一个对象后,就会把该用户指定为该对象的拥有者,并且该对象初始化为只有这个用户拥有访问它的权利。其他用户必须首先授予特权才可以访问该对象。优先权的构成成分包括:

(授权者,被授权者,对象,行为,可授权)

SQL 安全模型的两个主要支柱是特权和视图机制。它们为定义面向应用的安全策略提供了框架。

17.3.2 特权的授予和撤销

在 SQL 中,使用 GRANT 和 REVOKE 操作来管理特权。特权指的是特定的操作。而且仅仅被限制在一个表的特定属性之中。在我们的例子中,我们允许两个旅行社 Art 和 Zoe 通过以下语句来更新 Diary 表的某些部分:

```
GRANT SELECT, UPDATE (Day, Flight)
  ON TABLE Diary
  TO Art, Zoe
```

特权可以有选择地被撤销。

```
REVOKE UPDATE
  ON TABLE Diary
  FROM Art
```

更特别的是,在 SQL 中可以通过执行 GRANT 选项,来使得已经获得特权的用户授权给其他用户。例如:

```
GRANT SELECT
  ON TABLE Diary
  TO Art
  WITH GRANT OPTION
```

旅行社 Art 可以轮流地把 Diary 表的特权授予给 Zoe,

```
GRANT SELECT
```

```
ON TABLE Diary
TO Zoe
WITH GRANT OPTION
```

当 Diary 表的拥有者把授予给 Art 的特权撤销的时候, 所有由 Art 授予的特权也被迫撤销。因此撤销是级联的(cascade), 并且为了达到这样的目的, 要完成这样的操作所需的必要信息也通过数据库来保存。

应该明确的是: 一旦其他用户获得了访问数据的权限后, 即使数据的拥有者对原始数据仍然拥有某种控制, 但他们不能控制从这些数据中衍生出来的信息将如何被使用。你能够从一个表中读取数据, 也可以在没由获得对原始表写访问的权限下, 把读出的数据拷贝到另一个表中。

17.3.3 通过视图的访问控制

视图是从关系中派生出来的。SQL 语言中, 创建视图的操作应该遵循以下的格式:

```
CREATE VIEW view_name [ (column[ , column]...) ]
AS subquery
[ WITH CHECK OPTION];
```

在关系数据库中, 可以通过直接为基本表中的表项授予权限的操作来实现访问控制。然而, 许多安全的策略, 可以通过视图以及这些视图的特权来得到更好的表示。在视图定义中的子查询可以描述非常复杂的访问条件。作为一个简单的例子, 我们在例子关系 Diary 中构建一个包括所有公务旅行的视图。

```
CREATE VIEW business_trips AS
SELECT * FROM Diary
WHERE Status = 'business'
WITH CHECK OPTION;
```

通过视图, 访问控制能够被恰当地放在应用层中。数据库管理系统仅提供执行控制的一些工具。视图机制吸引我们的原因有以下几个原因:

- 视图非常灵活, 并且允许在贴近应用需求的描述层上定义一些访问控制策略。
- 视图能够增强上下文依赖和数据依赖的安全策略。
- 视图能够实施受控调用。
- 安全的视图可以代替安全标签。
- 数据可以很容易地重新分类。

面向应用的访问控制可以通过如下视图来表示,

```
CREATE VIEW Top_of_the_Class AS
SELECT * FROM Students WHERE Grade <
(SELECT Grade FROM Students WHERE Name = current_user());
```

该视图仅仅显示那些平均成绩比正在使用这个视图的学生的成绩还要低的学生, 或者使用以下视图来显示那些使用该视图的用户预定的旅行航线:

```
CREATE VIEW My_Journeys AS
SELECT * FROM Diary
WHERE Customer = current_user();
```

可以向数据库中添加访问控制表, 来实现自由访问控制。此时, 视图指的是关系。通过这种方式, 就可以表示基于组成员关系的访问控制, 也可以表示那些规定用户访问控制权限的授予和撤销的策略。

而且, 视图可以定义或者引用安全标签。在我们的例子中, 可以通过创建视图来把到达 Thule 的商务班机标记为机密的;

```
CREATE VIEW Flights ≡ CONFIDENTIAL AS
  SELECT * FROM Diary
  WHERE Destination = 'THU' AND Status = 'business' ;
```

通过视图来控制读取访问，除了能够正确地获取你的安全策略外，再也不会造成特殊的技术问题。但是当视图通过插入 (INSERT) 和更新 (UPDATE) 操作向数据库中写数据时，就应该另当别论了。首先，存在不可更新的视图，因为它们并没有包含有维护对应的基本关系所需要的信息。比如，一个视图中，如果没有包含强调基本关系的主键，那么该视图就是不能够更新的。其次，即使视图是可以更新的，仍然会留下一些有趣的安全问题。一个仅对 Diary 数据库有访问权利的旅行社可以通过 business_trips 视图来查看到下面的数据。

Name	Day	Flight	Status
Bob	Mon	YL011	business
Carol	Tue	BX201	business
Alice	Thu	FL9700	business

那是否应该允许旅行社通过下面的操作来更新视图呢？

```
UPDATE business_trips
  SET Status = 'private'
  WHERE Name = 'Alice' AND Day = 'Thu'
```

如果允许的话，那么 Alice 表项就要从视图中消失掉。事实上，在这种情况下，由于对视图的设置说明了检查 (CHECK) 选项，那么就不允许对视图的更新操作。如果对视图定义中包含了 (WITH CHECK OPTION)，那么更新 (UPDATE) 和插入 (INSERT) 都只能写入满足视图定义的数据库表项中。如果检查 (CHECK) 操作忽略了，那么就可能出现盲写 (blind write)。

在 SQL 安全模型中，视图不仅是一个对象，而且还可以被看成是一个程序。当视图是通过它的用户优先权而不是调用视图的用户具有的优先权来评价时，那么你就会找到另一种执行受控调用的方法。

对视图的访问条件必须在 SQL 的限制之内说明。如果这种方法太严格的话，那么就会用另外一种更富有表现力的语言书写的软件包 (存储程序)，来为数据库管理系统提供数据库的受控访问。用户将再次被授予软件包的执行权限，而此软件包与其用户的特权一起运行。

经验教训

计算机系统的任何一层都可以找到受控调用，该原则在微处理器和数据库管理系统中同样很有用。

至今为止，我们给出了视图的多个方面，这些方面使得视图成为一个很有用的安全机制。自然，视图机制也有它的不足之处。

- 访问控制可能变得相当复杂和缓慢。
- 必须检查视图的定义来保证正确性。它们真的得到我们期望的安全策略吗？
- 完整性和一致性，都不能自动达到，视图之间可能重复，也可能不能描述整个数据库。
- 数据库管理系统中，与安全相关的部分 (如 TCB) 可能变得相当大。

视图适合于通常的商务环境中。它们能够根据应用程序的需要裁剪，而不需要对数据库管理系统做任何修改。因此，视图的定义就是定义数据库结构的通用方法的一部分，从而它就可以最好地满足商务的需求。

但是当要确定哪个个体对某个数据项有访问权的时候，视图就可能变得很复杂了。因此，视图在这些情况下就不是很适合了，如：在认为要保护数据项而不是要控制用户的行为的时候。

17.4 统计数据库的安全

在本书中，至今为止都还没仔细研究过统计数据库产生的安全问题。统计数据库最显著的特征在于，它是通过对一张表中的一个属性(列)进行统计(聚集)查询来检索信息的。在 SQL 中，有以下一些聚集函数。

- COUNT：一列中值的个数。
- SUM：一列中值的总和。
- AVG：一列中值的平均值。
- MAX：一列中最大的值。
- MIN：一列中最小的值。

统计查询中的查询谓词(query predicate)指的是用来计算聚集的元组，而查询集(query set)是匹配查询谓词的元组。在内核中，统计数据库会出现以下安全问题。

- 数据库中包含一些数据项，它们通常是敏感数据。因此对数据项的访问就不会被允许。
- 对数据库的统计查询是允许的，但是由于它们自身的特点，这些查询读单个的数据项。

在这种情况下推导出信息是可能的。我们也会向你显示，仅仅通过管制访问控制请求是不够的。对于信息流，我们也将采取更加审慎的观点。在第 8 章和第 9 章中，机密性模型在试图阻止任何形式的信息流。在统计数据库中，必须有某种信息流从数据项流向它们的聚集。我们能做的也就是尽量把它们减少到可以接受的地步。

Name	Sex	Program	Units	Grade Ave.
Alma	F	MBA	8	63
Bill	M	CS	15	58
Carol	F	CS	16	70
Don	M	MIS	22	75
Errol	M	CS	8	66
Flora	F	MIS	16	81
Gala	F	MBA	23	68
Homer	M	CS	7	50
Igor	M	MIS	21	70

图 17-4 Students 关系表

图 17-4 的学生数据库，将在这节中提供许多例子。除了在单元数和平均分的单个表项上，其他所有属性上的统计性查询都是允许的。列上是不允许直接读取的。下面的统计性查询就是用来计算所有 MBA 学生的平均分数的。

```
Q1: SELECT AVG(Grade Ave. )
    FROM Students
    WHERE Programme = 'MBA'
```

在这个例子中的查询谓词是 Programme = 'MBA'。

17.4.1 聚集和推断

统计数据库安全性的两个重要的概念是聚集和推断。聚集指的是一种观察结果，也就是在

数据库中一组值计算出来的聚集的敏感度，与单个元素的敏感度不同。大多会遇到很多这样的情形：聚集的敏感度低于单个元素的敏感度。当聚集是从一个不那么敏感的商业数据中演化而来的敏感执行信息的时候，相反的情形也会发生。

聚集是数据库上的另一种关系，例如视图。因此，你可以使用本章提到的安全机制来控制对聚集的访问。但是，一个攻击者可以利用敏感度中的不同，来获得对更多敏感数据项的访问。推断问题指的是从一些不敏感的数据中，派生出一些敏感的数据。要考虑以下的一些攻击类型：

- 直接攻击：聚集是从小组样例中计算出来的，这样的话，私人的数据项就可能被泄漏出去。
- 间接攻击：这种攻击结合了与几个聚集关联的信息。
- 跟踪攻击：这是间接攻击的一种非常有效的类型。
- 线性系统的漏洞攻击：这种攻击又比跟踪攻击进了一步，它采用查询集中的算术关系，来构建等式，以产生期望的信息。

17.4.2 跟踪攻击

现在我们将说明如何利用统计性查询从学生关系数据库中，推导出敏感的数据。假设我们知道 Carol 是计算机系的一名女生，通过以下查询：

```
Q1: SELECT COUNT(*)
     FROM Students
     WHERE Sex = 'F' AND Programme = 'CS'
```

```
Q2: SELECT AVG(Grade Ave.)
     FROM Students
     WHERE Sex = 'F' AND Programme = 'CS'
```

从 Q1 中，我们知道如果数据库中的计算机系只有一个女生，那么由 Q2 返回的值 70 就是她的平均成绩。现在问题出现了，在查询的标准中，我们定义的集合可以只包含一个数据元素。因此只有当统计性查询覆盖了一个足够大的子集时，它才被允许执行。但是我们可以忽略选择标准，仅仅查询补集，在对整个数据库的查询结果和对我们感兴趣的补集结果的差异中，得到一致的结果。因此你不仅需要要求查询的元组集合，而且它的补集要相当的大。

遗憾的是，即使这样也不够好。假设每个查询的集合和它的补集必须包含至少三个元素。查询顺序如下：

```
Q3: SELECT COUNT(*)
     FROM Students
     WHERE Programme = 'CS'
```

```
Q4: SELECT COUNT(*)
     FROM Students
     WHERE Programme = 'CS' AND Sex = 'M'
```

```
Q5: SELECT AVG(Grade Ave.)
     FROM Students
     WHERE Programme = 'CS'
```

```
Q6: SELECT AVG(Grade Ave.)
     FROM Students
     WHERE Programme = 'CS' AND Sex = 'M'
```

返回的值分别是 Q3: 4, Q4: 3, Q5: 61, Q6: 58。所有查询都考虑到了元组集合是充分大的，所以它们是不被禁止的。但是，通过结合四个结果，我们计算出 Carol 的平均成绩是 $4 \times 61 - 3 \times 58 = 70$ 。

你可能会认为这个例子很巧合, 因为 Carol 是计算机系中唯一一个女生, 因此可以构造出查询集。现在我们将向你展示用系统的方式, 如何来设置攻击。首先, 我们需要一个追踪者。

允许对单个元组追踪其信息的查询谓词 T 被称为该元组的单个追踪者(individual tracker)。

一个通用的追踪者(general tracker)是一个谓词, 它能够用来找到任何不允许查询的答案。

假设 T 是一个普通的追踪者, R 是一个谓词, 这个谓词唯一地标志我们想要获取的元组 r 。在我们的例子中, 谓词是 $Name = 'Carol'$ 。我们使用谓词 $R \vee T$ 和 $R \vee \text{NOT}(T)$ 对数据库作出两条查询。我们的目标 r 是被两个查询使用到的唯一元组。若要保证两个查询都是允许的, 那么我们可以选择 T 使查询集及其补集都足够大, 从而允许查询。整个数据库的最终查询结果会向我们提供所有的数据来完成攻击。在我们的例子中:

```
Sex = 'F' AND Programme = 'CS'
```

是对 Carol 的单个追踪者, $Programme = 'MIS'$ 是许多普通追踪者中的一个。我们继续以下的查询:

```
Q7: SELECT SUM(Units)
     FROM Students
     WHERE Name = 'Carol' OR Programme = 'MIS'
```

```
Q8: SELECT SUM(Units)
     FROM Students
     WHERE Name = 'Carol' OR NOT (Programme = 'MIS')
```

```
Q9: SELECT SUM(Units)
     FROM Students
```

并得到 Q7: 75, Q8: 77 和 Q9: 136。因此 Carol 一定已经得到了 $(75 + 77) - 136 = 16$ 个学分。经验表明: 几乎所有的统计数据库都有一个普通的追踪者。

17.4.3 对策

对于统计推理攻击的分析已经在早期数据库安全方面的书籍中介绍很多了。自从那时起, 研究人员就把精力转移到其他领域了, 这与其说是因为找到了完整的而严密的解决方法, 还不如说是因为他们不得不承认, 抵制推理攻击的对策还有很多限制。考虑到这些限制, 那么我们对于推理问题真正能做什么呢。

首先, 你必须明确地保护好敏感地信息。这也就暗示着你知道哪些信息是敏感的信息, 哪些信息有可能成为敏感的信息。而且你还要知道应该怎样获取信息。至少, 你应该在得到统计查询结果之前能够检查查询结果的规模。

其次, 你应该能够伪装数据。你可能会随机地交换数据库中的表项, 这样做的结果是, 即使统计性查询仍然是正确的, 但是对于单个的查询, 其结果就可能是错误的了。也就是说, 可以向查询结果中添加一些随机的混乱信息, 这样返回来的值就会更接近真实的值, 但是仍然不会非常准确。这些技术的一种缺陷, 就是会减少精度和可用性。你可能不希望在医药数据库中随机地交换各种数据值。

可以通过精心设计的数据库模式, 来解决一些聚集问题(Lunt, 1989)。对数据库结构的静态分析可能会揭露出属性之间的敏感关系。这些属性被分派到独立的表中。仅仅对某一数据表有访问权的用户, 就不能再把各个属性之间的关系关联起来了。当然, 那些拥有对所有表的访问权的用户, 仍然能够得到相关的信息, 但是作为一个数据库管理员来说, 在分配优先权的时候, 就可能会更加精确了。在我们的例子中, 名字和学习成绩之间的关系是敏感的信息。我们可以通

过学生身份号码，用两个表来代替 Students 表了(图 17-5)。

Name	ID	ID	Sex	Programme	Units	Grade Ave.
Alma	B13	B13	F	MBA	8	63
Bill	C25	C25	M	CS	15	58
Carol	C23	C23	F	CS	16	70
Don	M38	M38	M	MIS	22	75
Errol	C12	C12	M	CS	8	66
Flora	M22	M22	F	MIS	16	81
Gala	B36	B36	F	MBA	23	68
Homer	C10	C10	M	CS	7	50
Igor	M20	M20	M	MIS	21	70

图 17-5 把 Students 分成两个表

现在第一个表可以在安全级方面设置得更高，从而只有授权的用户才能把姓名和学分联系起来。

最后，我们会发现推理问题不是由单个的查询引起的，而是通过将几个查询巧妙地集合起来而引起的。因此你可能也会追踪到底用户知道些什么。这可能会给出最好的安全性，但是这也可能是代价最昂贵的选择。用户的行为可以记录在一个审计日志中，如果检查到一个可疑的查询序列，你也可以执行查询分析(query analysis)。当然，首先，你不得不搞清楚的是，可疑的查询行为由哪些查询行为构成。为了加强保护，查询分析就必须考虑两个用户，或是一组用户，他们同时知道什么。

17.5 操作系统的完整性

当你从操作系统的角度看待数据库的时候，你便会发现许多的用于维护数据表项操作系统进程，以及保存数据项的内存资源。数据库管理系统在很多方面都和操作系统有相同的职责。它必须能防止用户之间相互干扰，也要能防止和数据库管理系统相互冲突。

如果你不想花费过多的精力，你可以把这些任务交给操作系统。在这种方式里，数据库管理系统作为操作系统的进程集工作。数据库中有许多进程来完成有通用的数据库管理系统大的进程，并且数据库的每个用户也对应于一个独立的操作系统的进程(图 17-6)。现在操作系统能够区分不同的用户，并且如果你将自己的文件存储到数据库管理系统中的话，那么操作系统就能够执行所有的访问控制了。数据库系统只要把用户的查询翻译成操作系统的能够理解的操作就行了。

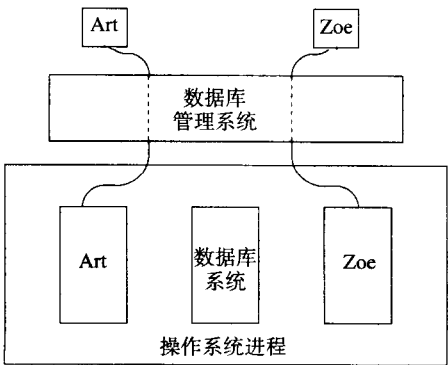


图 17-6 操作系统下的数据库用户隔离

把每一个操作系统的进程分派给每个数据库的用户非常耗内存资源,而且也不会覆盖到大部分的用户。因此,你需要的是能够处理几个用户请求的进程(图 17-7)。你可能会节约内存,但是现在访问控制方面的责任也集中取决于数据库管理系统。在数据库中存储对象也需要相似的考虑。如果对象过小,那么每个对象都拥有一个独立的文件,就会显得非常的浪费。一旦操作系统失去了对数据库用户的访问控制功能,那么你就能够自由地在操作系统文件中收集一些数据库对象。

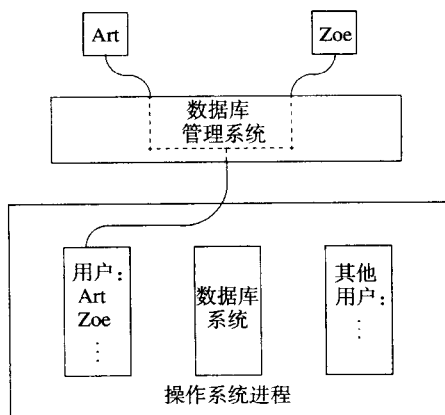


图 17-7 数据库管理系统下的数据库用户隔离

17.6 隐私

许多组织出于法律原因都会存储各自用户的个人数据,比如:姓名、地址、信用卡卡号、个人偏好,或其他用户的习惯。这些数据一般都是被法律保护的。维护数据一定要遵从法律法规的限制。

为了保护个人数据,国际上推荐使用保护隐私的方针(Guidelines on the Protection of Privacy and Transborder Flows of Personal Data, OECD)。这个方针陈述了八条保护原则,在这些原则中,数据主体都是数据引用的个体,而数据的控制者,则是指的是维护数据库的人。

1. 收集限制原则:应该限制对个人数据的收集,数据应该通过法律和合法方式得到,并且得到数据主体的认可和同意。

2. 数据质量原则:单个数据应该要与一些目的联系,如:数据要可使用,要精确,完整,而且要及时。

3. 目的说明书原则:应该说明个人数据收集的目的。

4. 使用限制原则:除特定目的不得批露和使用个人数据,否则必须经由数据主体批准或法律授权许可。

5. 安全防卫原则:应该通过合理的安全访问方式,来保护个人的数据。

6. 开放性原则:与个人数据相关的进展、实践和策略的开放性都应该有一个通用的政策。

7. 个体参与原则:单个个体应该拥有确认以下信息的权利:控制者是否拥有与他(她)相关的数据,能否检索与他(她)相关的数据,在请求得到否定的时候,能否被告知原因。

8. 问责性原则:数据的控制者应该对他实施的一些措施担负责任,而这些措施可能都是用来实施上述原则的。

处理个人数据以及像这样的数据自由移动的另一个重要的文档是欧盟联盟标准^①。欧盟指令并不是法律，但是必须由欧盟成员国放入国家法律条文中。该标准非常强调用户的批示。用户必须参加进来说明他们同意托管保存自己的数据。或采用另一种策略：数据主体应该显式地陈述数据不能被保留。

在美国有多项法律都对数据保护进行了阐述，比如，健康保险携带与责任法 (Health Insurance Portability and Accountability Act of 1996, HIPPA)，就是一部保护病人的医疗记录以及提供给健康机构、医生、医院和健康照看人员的病人健康状况信息。这部法律在 2003 年生效，它对通用情况下数据库安全和信息技术安全都产生了巨大的影响。

17.7 深层阅读

你可以参看其他书籍进一步了解关系数据模型。关于数据库安全方面的许多资料都已经收集并编撰在 *Castano et al (1994)* 的著作中。更早的但是也很有用的关于数据库安全的书包括 *Denning (1982)* 的著作。这本书为统计数据库的安全提供了一个很好的参考价值。关于这方面更多的有用的资料，可以在 *Lunt (1989)* 的著作中查到。所有主要的数据库卖主都把他们的产品用网页附带信息的形式保留起来，同时也对他们的数据安全做了很好的介绍。

17.8 习题

练习 17.1 假设有一个账户数据库，它的记录是(客户名，账户号，余额，信用率，以及以下一些用户类型：客户，职员，管理员。试定义一个访问结构，比如通过视图，来实现如下一些功能：

- 客户可以读取他们自己的账户信息。
- 职员可以读取除了信用率以外所有的字段信息，也可以更新所有用户的余额。
- 管理员可以创建新的记录，读取所有的字段信息以及为所有用户更新他们的信用率。

练习 17.2 假设有一个学生记录的数据库，其中包含了学生的姓名，学生的各个课程的分。初稿提供的视图显示了所有学生的课程，这些学生的试卷都还没有被给出最后的分数。那么是否这些视图就应该定义成 WITH CHECK OPTION 呢？对决定是否使用 CHECK OPTION 给出通用的标准。

练习 17.3 所有的在 Students 关系(图 17-4)上的统计性查询在其查询结果中都至少要包含三个元组。只有在属性 Grade Ave 上的 AVG 查询才是允许的。试找出一个新的通用的追踪者，并且构建一个对 Homer 的平均分数的跟踪攻击。

练习 17.4 在数据库系统安全中，你可能已经看过了放置在美国应用层上的安全控制的实例。那么这种方法的问题是什么呢？

练习 17.5 假设在一个数据库系统中，一些聚集被放在比产生聚集的数据项更高的敏感层上。拥有访问数据项权限的用户，就可能通过单个地访问数据项，来计算出聚集。那么你怎样防御这种攻击呢？

练习 17.6 如果给你一个数据库，在这个数据库中，对于表中的每一行的访问权限都是单独定义的。访问控制应该使用操作系统的安全机制。如果通过把数据库对象存储在操作系统中的文件中这种方式来决定设计，那么这样的设计决策的结果是什么呢？(作为一种标准，试考虑一下操作系统为每个文件和拥有 1000 万个记录的数据，仅仅使用 100 个管理数据的字节。)这是一种可行的方案吗？

① 是 1995 年 10 月 24 日由欧洲议会和理事会共同通过的 95/46/EC 欧盟标准 95/46/EC。

参考文献

- Abu El-Asa, Ahmed, Martin Aeberhard, Frank J. Furrer, Ian Gardiner-Smith, and David Kohn (2002) *Our PKI-Experience*. SYSLOGIC Press, Birmensdorf, Switzerland.
- Adams, Anne and Martina Angela Sasse (1999) Users are not the enemy. *Communications of the ACM*, 42(12):40–46.
- Alberts, C. and A. Dorofee (2003) *Managing Information Security Risks*. Pearson Education Inc., Boston, MA.
- Alpern, B. and S. Schneider (1985) Defining liveness. *Information Processing Letters*, 21(4):181–185.
- Amoroso, E. (1994) *Fundamentals of Computer Security Technology*. Prentice Hall, Englewood Cliffs, NJ.
- Anderson, J. (October 1972) Computer security technology planning study. Technical Report 73–51, US Air Force Electronic Systems Technical Report.
- Anderson, Ross (2001) *Security Engineering*. John Wiley & Sons, New York.
- Arbaugh, William A., William L. Fithen and John McHugh (2000) Windows of vulnerability: a case study analysis. *IEEE Computer*, 33(12):52–59.
- Arceneaux, J. (October 1996) Experiences in the art of security. *Security Audit & Control-Review*, 14(4):12–16.
- Arsenault, Alfred W. and Sean Turner (November 2000) Internet X.509 Public Key Infrastructure – Roadmap. Internet Draft draft-ietf-pkix-roadmap-06.txt. <http://www.ietf.org>.
- Ashcraft, Ken and Dawson Engler (2002) Using programmer-written compiler extensions to catch security holes. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 143–159.
- Ashley, Paul M. and Mark Vandenwauver (1999) *Practical Intranet Security*. Kluwer Academic Publishers.
- Atkins, D., P. Buis, C. Hare, R. Kelley, C. Nachenberg, A.B. Nelson, P. Phillips, T. Ritchey, T. Sheldon and J. Snyder (1997) *Internet Security*. New Riders, Indianapolis, IN, 2nd edition.
- Aura, Tuomas (2003) Cryptographically generated addresses (cga). In W. Mao C. Boyd, editor, *Proceedings ISC'03, LNCS 2851*, pages 29–43. Springer-Verlag.
- Aura, Tuomas, Michael Roe and Jari Arkko (December 2002) Security of Internet location management. In *Proceedings of the 18th Annual Computer Security Applications Conference*, pages 78–87.
- Baker, D.B. (September 1996) Fortresses built upon sand. In *Proceedings of the New Security Paradigms Workshop*, pages 148–153.
- Bejtlich, Richard (2000) Interpreting network traffic: a network intrusion detector's look at suspicious events. In *Proceedings of the 12th Annual Computer Security Incidence Handling Conference*, Chicago.
- Bell, D.E. and L.J. LaPadula (1996) MITRE technical report 2547 (secure computer system): Volume II. *Journal of Computer Security*, 4(2/3):239–263.
- Bell, David and Leonard LaPadula (1975) Secure computer system: unified exposition and Multics implementation. Technical Report ESD-TR-75-306, The MITRE Corporation, Bedford, MA, July 1975.
- Bellare, Mihir and Phillip Rogaway (1994) Entity authentication and key distribution.

- In D.R. Stinson, editor, *Advances in Cryptology-CRYPTO' 93*, LNCS 773, pages 232-249. Springer-Verlag.
- Bellare, Mihir and Phillip Rogaway (1995) Optimal asymmetric encryption padding. In A. De Santis, editor, *Advances in Cryptology - Proceedings Eurocrypt 94*, LNCS 950, pages 92-111. Springer-Verlag.
- Bellare, Mihir, Ran Canetti and Hugo Krawczyk (1996) Keyed hash functions and message authentication. In N. Koblitz, editor, *Advances in Cryptology - Proceedings Crypto 96*, LNCS 1109, pages 1-15. Springer-Verlag.
- Bellovin, S. (1989) Security problems in the TCP/IP protocol suite. *ACM Computer Communications Review*, 19(2):32-48, April.
- Bellovin, Steve M. and Michael Merritt (1990) Limitations of the Kerberos Authentication System. *ACM Computer Communications Review*, 20(5):119-132.
- Bellovin, Steve M. and Michael Merritt (1992) Encrypted key exchange: password-based protocols secure against dictionary attacks. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, pages 72-84.
- Berstis, D. et al. (1978) IBM System/38 addressing and authorization. Technical Report GS80-0237, IBM System/38 Technical Development.
- Besson, Frédéric, Tomasz Blanc, Cédric Fournet and Andrew D. Gordon (2004) From stack inspection to access control: a security analysis for libraries. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop*, pages 61-75. IEEE Computer Society.
- Biba, K.J. (1977) Integrity consideration for secure computer systems. Technical Report ESD-TR-76-372, MTR-3153, The MITRE Corporation, Bedford, MA, April.
- Bieber, P., J. Cazin, P. Girard, J.-L. Lanet, V. Wiels and G. Zanon (2000) Checking secure interactions of smart card applets. In F. Cuppens et al., editors, *ESORICS 2000*, LNCS 1895, pages 1-16. Springer-Verlag.
- Bird, R., I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva and M. Yung (1995) The KryptoKnight family of light-weight protocols for authentication and key distribution. *IEEE/ACM Transactions on Networking*, 3(1):31-41, February.
- Blakely, B. (1996) The emperor's old armour. In *Proceedings of the New Security Paradigms Workshop*, pages 2-16, September.
- Blaze, Matt, Joan Feigenbaum and Jack Lacy (1996) Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164-173.
- Bleichenbacher, Daniel (1998) A chosen ciphertext attack against protocols based on RSA encryption standard PKCS # 1. In H. Krawczyk, editor, *Advances in Cryptology - Crypto 98*, LNCS 1462, pages 1-12. Springer-Verlag.
- Brewer, D.F.C. and M.J. Nash (1989) The Chinese Wall security policy. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 206-214.
- Brickell, Ernie, Jan Camenisch and Liqun Chen (2004) Direct anonymous authentication. In B. Pfizmann et al., editors, *Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 132-145. ACM Press.
- Brown, Keith (2000) *Programming Windows Security*. Addison-Wesley.
- BSI (2003) IT baseline protection manual. Technical report, Bundesamt für Sicherheit in der Informationstechnik, Bonn, Germany, October.
- Camenisch, Jan (2004) Better privacy for Trusted Computing Platforms. In P. Samarati et al., editors, *ESORICS 2004*, LNCS 3193, pages 73-88. Springer-Verlag.
- Canadian System Security Center (1993) *The Canadian Trusted Computer Product Evaluation Criteria*, Version 3.0e.
- Cardelli, Luca (1997) Type systems. In *Handbook of Computer Science and Engineering*, Chapter 103. CRC Press.

- Castano, S., M. Fugini, G. Martella and P. Samarati (1994) *Database Security*. Addison-Wesley, Reading, MA.
- CCIB (2004a) *Common Criteria for Information Technology Security Evaluation*, Version 2.2.
- CCIB (2004b) *Common Methodology for Information Technology Security Evaluation – Part 2: Evaluation Methodology*, Version 2.2.
- CCITT (1988) *The Directory – Overview of Concepts, Models and Services*, CCITT Rec X.500.
- Cheswick, William R., Steven M. Bellovin and Aviel D. Rubin. (2003) *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley, Reading, MA, 2nd edition.
- Chokhani, S. (1992) Trusted product evaluations. *Communications of the ACM*, 35(7):64–76, July.
- Clark, D.R. and D.R. Wilson (1987) A comparison of commercial and military computer security policies. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 184–194.
- CNSS (2003) *National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information*, June. CNSS Policy No. 15, Fact Sheet No. 1.
- Commission of the European Communities (1991) *Information Technology Security Evaluation Criteria (ITSEC)*, Version 1.2.
- Commission of the European Communities (1993) *Information Technology Security Evaluation Manual (ITSEM)*.
- Corbato, F.J. (1991) On building systems that will fail. *Communications of the ACM*, 34(9):72–81, September.
- Cowan, Crispian, Calton Pu, Dave Maier, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, Qian Zhang and Heather Hinton (1998) StackGuard: automatic adaptive detection and prevention of buffer-overflow attacks. In *Proceedings of the 7th USENIX Security Symposium*, pages 63–78.
- Curry, D.A. (1992) *Unix System Security*. Addison-Wesley, Reading, MA.
- Daemen, Joan and Vincent Rijmen (1999) AES proposal: Rijndael. Technical report, September, AES Algorithm Submission.
- Date, C.J. (1990) *An Introduction to Database Systems – Volume I*. Addison-Wesley, Reading, MA, 5th edition.
- Daugman, John (1993) High confidence visual recognition of persons by a test of statistical independence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1148–1161.
- Denning, D.E., T.F. Lunt, R.R. Schell, W.R. Shockley and M. Heckman (1988) The SeaView Security Model. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, pages 218–233.
- Denning, Dorothy E. (1982) *Cryptography and Security*. Addison-Wesley, Reading, MA.
- Denning, Dorothy E. and Giovanni M. Sacco (1981) Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536.
- Diffie, Whitfield, Paul C. van Oorschot and Michael J. Wiener (1992) Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125.
- Diffie, Whitfield and Martin E. Hellman (1976) New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654.
- Dobbertin, Hans (1996) Cryptanalysis of MD4. In D. Gollmann, editor, *Fast Software Encryption*, LNCS 1039, pages 53–69. Springer-Verlag.

- Dreyfuss, S. (1997) *Underground*. Reed Books.
- Eichin, M.W. and J.A. Rochlis (1989) With microscope and tweezers: an analysis of the Internet virus of November 1988. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 326–343.
- ElGamal, Tahir (1985) A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472.
- Ellis, J.H. (1970) The possibility of non-secret encryption. Technical report, CESC, January <http://www.cesg.gov.uk/about/nsecret/home.htm>.
- Ellison, Carl M., Bill Frantz, Butler Lampson, Ron Rivest, Brian M. Thomas and Tatu Ylonen (1999) *SPKI Certificate Theory*, September, RFC 2693.
- England, P., B. Lampson, J. Manferdelli, M. Peinado and B. Willman (2003) A trusted open platform. *IEEE Computer*, 36(7):55–62.
- Erlingsson, Úlfar and Fred B. Schneider (2000) IRM enforcement of Java stack inspection. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 246–255.
- European Computer Manufacturers Association (1993) Secure information processing versus the concept of product evaluation. Technical Report ECMA TR/64, December.
- Feldmeier, D.C. and P.R. Karn (1990) UNIX password security – ten years later. In *Advances in Cryptology – CRYPTO'89, LNCS 435*, pages 44–63. Springer-Verlag.
- Ferbrache, D. and G. Shearer (1992) *UNIX Installation Security and Integrity*. Blackwell Scientific Publications, Oxford.
- Fluhrer, Scott R., Itsik Mantin and Adi Shamir (2001) Weaknesses in the key scheduling algorithm of RC4. In Amr M. Youssef and S. Vaudenay, editors, *Proceedings SAC 2001, LNCS 2259*, pages 1–24. Springer-Verlag.
- Ford, W. (1994) *Computer Communications Security* Prentice Hall, Englewood Cliffs, NJ.
- Foster, James C., (2005) *Buffer Overflow Attacks*. Syngress Publishing, Rockland, MA.
- Fournet, Cédric and Andrew D. Gordon (2003) Stack inspection: theory and variants. *ACM Transactions on Programming Languages and Systems*, 25(3):360–399, May.
- Framer, D. and E.H. Spafford (1990) The COPS security checker system. In *The Summer Usenix Conference*, Anaheim, CA.
- Garfinkel, Simson, Gene Spafford and Alan Schwartz (2003) *Practical Unix & Internet Security*. O'Reilly & Associates, Sebastopol, CA, 3rd edition.
- Gasser, M. (1988) *Building a Secure Computer System*. Van Nostrand Reinhold, New York. <http://www.acsac.org/secshelf/book002.html>.
- Gasser, M. (1990) The role of naming in secure distributed systems. In *Proceedings of the CS'90 Symposium on Computer Security*, pages 97–109, Rome, Italy, November.
- Gasser, M., A. Goldstein, C. Kaufman and B. Lampson (1989) The digital distributed system security architecture. In *Proceedings of the 1989 National Computer Security Conference*.
- Gligor, V.D. (1984) A note on denial of service in operating systems. *IEEE Transactions on Software Engineering*, 10(3):320–324, May.
- Goguen, J.A. and J. Meseguer (1982) Security policies and security models. In *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, pages 11–20.
- Gollmann, Dieter (2003) Authentication by correspondence. *IEEE Journal on Selected Areas in Communications*, 21(1):88–95, January.
- Gong, Li (1999) *Inside Java 2 Platform Security*. Addison-Wesley, Reading, MA.
- Gong, Li, Mary Dageforde and Gary W. Ellison (2003) *Inside Java 2 Platform Security*. Addison-Wesley, Reading, MA, 2nd edition.

- Govindavajhala, Sudhakar and Andrew W. Appel (2003) Using memory errors to attack a virtual machine. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, page 154–165.
- Graff, Mark G. and Kenneth R. van Wyk (2003) *Secure Coding*. O'Reilly & Associates.
- Granville, Andrew (2005) It is easy to determine whether a given integer is prime. *Bulletin (New Series) of the American Mathematical Society*, 42(1):338.
- Grover, D. (ed.) (1992) *The Protection of Computer Software – its Technology and Applications*. Cambridge University Press, Cambridge, 2nd edition.
- Hadfield, L., D. Hatter and D. Bixler (1997) *Windows NT Server 4 Security Handbook*. Que Corporation, Indianapolis, IN.
- Halevi, Shai and Hugo Krawczyk (1999) Public-key cryptography and password protocols. *ACM Transactions on Information and System Security*, 2(3):230–268, August.
- Hallem, Seth, Benjamin Chelf, Yichen Xie and Dawson Engler (2002) A system and language for building system-specific, static analyses. In *Proceedings of PLDI 2002, June 17–19, 2002, Berlin, Germany*, pages 69–82. ACM Press.
- Hansen, P. Brinch (1973) *Operating Systems Principles*. Prentice Hall, Englewood Cliffs, NJ.
- Harrison, M.A., W.L. Ruzzo and J.D. Ullman (1976) Protection in operating systems. *Communications of the ACM*, 19(8):461–471, August.
- Hellman, M.E. (1980) A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406.
- Hennessy, J. and D. Patterson (2002) *Computer Architecture – A Quantitative Approach*. Morgan Kaufmann, San Mateo, CA, 3rd edition.
- Hogben, Giles, Tom Jackson and Marc Wilikens (2002) A fully compliant research implementation of the P3P standard. In D. Gollmann *et al.*, editors, *Computer Security – ESORICS 2002, LNCS 2502*, pages 104–120. Springer-Verlag.
- Housley, Russell, Tim Polk, Warwick Ford and David Solo (2002) *Internet X.509 Public Key Infrastructure – Certificate and Certificate Revocation List (CRL) Profile*, April, RFC 3280.
- Howard, Michael and David LeBlanc (2002) *Writing Secure Code*. Microsoft Press, Redmond, WA, 2nd edition.
- Howard, Michael, Jon Pincus and Jeanette M. Wing (2003) Measuring relative attack surfaces. Technical Report CMU-CS-03-169, Carnegie Mellon University, Pittsburgh, PA.
- International Organization for Standardization (1989) *Basic Reference Model for Open Systems Interconnection (OSI) Part 2: Security Architecture*. Geneva, Switzerland.
- International Organization for Standardization (1991) *Information technology – Security Techniques – Entity Authentication Mechanisms; Part 1: General Model*. Geneva, Switzerland, September. ISO/IEC 9798-1, Second Edition.
- International Organization for Standardization (1992) *ISO/IEC 9075: Information Technology – Database Languages – SQL*. Geneva, Switzerland.
- International Organization for Standardization (1997) *Information technology – Open Systems Interconnection – The Directory-Authentication Framework*. Geneva, Switzerland, June. ISO/IEC 9594-8 – ITU-T Rec X.509 (1997 E).
- International Organization for Standardization (2001) *Information Technology – Code of Practice for Information Security Management*. Geneva, Switzerland.
- Kahn, D. (1967) *The Codebreakers*. Macmillan Publishing Company, New York.
- Kang, M.H., A.P. Moore and I.S. Moskowitz (1998) Design and assurance strategy for the NRL pump. *IEEE Computer*, 31(4):56–64, April.
- Karger, P.A. (1991) Implementing commercial data integrity with secure capabilities. In

- Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*, pages 130–139.
- Karger, P.A., M.E. Zurko, D.W. Bonin, A.H. Mason and C.E. Kahn (1990) A VMM security kernel for the VAX architecture. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, pages 1–19.
- Karger, Paul A. and Roger R. Schell (2002) Thirty years later: lessons from the Multics security evaluation. In *Proceedings of ACSAC 2002*, pages 119–148.
- Kohl, J. and C. Neumann (1993) *The Kerberos Network Authentication Service (V5)*, September, Internet RFC 1510.
- Krawczyk, Hugo, Mihir Bellare and Ran Canetti (1997) HMAC: keyed-hashing for message authentication. Technical report, February, RFC 2104.
- La Macchia, Brian A., Sebastian Lange, Matthew Lyons, Rudi Martin and Kevin T. Price (2002) *.NET Framework Security*. Addison-Wesley Professional, Boston, MA.
- Lamport, L. (1979) Constructing digital signatures from a one way function. Technical Report CSL-98, SRI International Computer Science Laboratory, October.
- Lamport, L. (1985) Checking secure interactions of smart card applets. In M.W. Alford *et al.*, editors, *Distributed Systems: Methods and Tools for Specification: An Advanced Course*, LNCS 190, pages 119–130. Springer-Verlag.
- Lampson, B. (1974) Protection. *ACM Operating Systems Reviews*, 8(1):18–24, January.
- Lampson, Butler, Martin Abadi, Michael Burrows and Edward Wobber (1992) Authentication in distributed systems: theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, November.
- Landwehr, C. (1983) The best available technologies for computer security. *IEEE Computer*, 16(7):86–100.
- Lang, Ulrich and Rudolf Schreiner (2002) *Developing Secure Distributed Systems with CORBA*. Artech House, Norwood, MA.
- Laprie, J.-C. (1992) *Basic Concepts and Terminology*. Springer-Verlag, Vienna.
- Lee, Ruby B., David K. Karig, John P. McGregor and Zhijie Shi (2003) Enlisting hardware architecture to thwart malicious code injection. In *Proceedings of the International Conference on Security in Pervasive Computing (SPC-2003)*, LNCS 2802, pages 237–252. Springer-Verlag.
- Lee, T.M.P. (1991) Using mandatory integrity to enforce “commercial” security. In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*, pages 140–146.
- Lipton, R.J. and L. Snyder (1978) On synchronization and security. In R.D. Demillo *et al.*, editors, *Foundations of Secure Computation*, pages 367–385. Academic Press, New York.
- Lunt, T.F. (1989) Aggregation and inference: facts and fallacies. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 102–109.
- Lunt, T.F., D.E. Denning, R.R. Schell, M. Heckman and W.R. Shockley (1990) The seaview security model. *IEEE Transactions on Software Engineering*, 16(6):593–607.
- MacKenzie, D. and G. Pottinger (1997) Mathematics, technology, and trust: formal verification, computer security, and the U.S. Military. *IEEE Annals of the History of Computing*, 19(3):41–59.
- Manger, James (2001) A chosen ciphertext attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as standardized in PKCS # 1 v2.0. In J. Kilian, editor, *Advances in Cryptology – Crypto 2001*, LNCS 2139, pages 230–238. Springer-Verlag.
- Matsumoto, Tsutomu (2002) Gummy and conductive silicon rubber fingers – importance of vulnerability analysis. In Y. Zheng, editor, *Advances in Cryptology – Asiacrypt*

- 2002, *Queenstown, New Zealand, LNCS 2501*, pages 574–575. Springer-Verlag.
- McGraw, G. and E.W. Felten (1997) *Java Security*. John Wiley & Sons, New York.
- McGraw, Gary (2004) *Exploiting Software: How to Break Code*. Addison-Wesley.
- McLean, J. (1987) Reasoning about security models. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 123–131.
- McLean, J. (January 1990) The specification and modeling of computer security. *IEEE Computer*, 23(1):9–16.
- McLean, J. (1994) Security models. In J. Marciniak, editor, *Encyclopedia of Software Engineering*. John Wiley & Sons, New York.
- Menezes, Alfred J., Paul C. van Oorschot and Scott A. Vanstone (1997) *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL.
- Miller, B.F., L. Frederiksen and B. So (December 1990) An empirical study of the reliability of Unix utilities. *Communications of the ACM*, 33(12):32–44.
- Miller, S.P., B.C. Neuman, J.I. Schiller and J.H. Saltzer (1987) Section E.2.1: Kerberos authentication and authorization system. Technical report, MIT Project Athena, Cambridge, MA.
- Mitchell, Christopher J. and Paolo S. Pagliusi (2003) Is entity authentication necessary? In B. Christiansen *et al.*, editors, *Security Protocols, 10th International Workshop, Cambridge, LNCS 2845*, pages 20–33. Springer-Verlag.
- Mitnick, Kevin D. and William L. Simon (2002) *The Art of Deception*. John Wiley & Sons, Indianapolis, IN.
- Morris, R. and K. Thompson (November 1979) Password security: a case history. *Communications of the ACM*, 22(11):594–597.
- Morris, R.T. (February 1985) A weakness in the 4.2BSD Unix TCP/IP Software. Bell Labs Computer Science Technical Report.
- National Institute of Standards and Technology & National Security Agency (1992) *Federal Criteria for Information Technology Security*, Version 1.0.
- NCSC (1987) *Trusted Network Interpretation*, NCSC-TG-005, Version 1.0.
- NCSC (April 1991) *Trusted Database Management System Interpretation*, NCSC-TG-021.
- Necula, George C. (January 1997) Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '97)*, pages 106–119, Paris.
- Needham, R.M. (1992) Later developments at Cambridge: Titan, CAP, and the Cambridge Ring. *Annals of the History of Computing*, 14(4):57.
- Needham, R.M. and M.D. Schroeder (1978) Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999.
- Nelson, R.P. (1988) *The 80386 Book*. Microsoft Press.
- Neumann, J. von, (1993) First draft of a report on the EDVAC (M.D. Godfrey (ed.)). *Annals of the History of Computing*, 15(4):27–75.
- Niemi, Valtteri and Kaisa Nyberg (2003) *UMTS Security*. John Wiley & Sons, Chichester.
- Organick, E.I. (1972) *The Multics System: An Examination of Its Structure*. MIT Press, Cambridge, MA.
- Park, J.S. (1995) *AS/400 Security in a Client/Server Environment*. John Wiley & Sons, New York.
- Pfleeger, C.P. and S. Lawrence Pfleeger (2003) *Security in Computing*. Prentice Hall, Englewood Cliffs, NJ, 3rd edition.
- Preneel, B., B.B. Van Rompay, S.B. Örs, A. Biryukov, L. Granboulan, E. Dottax, M. Dichtl, M. Schafheutle, P. Serf, S. Pyka, E. Biham, E. Barkan, O. Dunkelman, J. Stolin, M. Ciet, J.-J. Quisquater, F. Sica, H. Raddum and M. Parker (February 2003)

- Performance of optimized implementations of the NESSIE primitives. Technical report, Version 2.0. <http://www.cryptoneessie.org>.
- Ptacek, Thomas H., and Timothy N. Newsham (1998) Insertion, evasion, and denial of service: eluding network intrusion detection. Technical report, Secure Networks, Inc.
- Rivest, Ron, Adi Shamir and L. Adleman (1978) A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.
- Russel, D. and G.T. Gangemi Sr (1991) *Computer Security Basics*. O'Reilly & Associates, Sebastopol, CA.
- Saltzer, J.H. (1974) Protection and the control of information sharing in Multics. *Communications of the ACM*, 17:388–402.
- Samalin, S. (1997) *Secure Unix*. McGraw-Hill.
- Sandhu, R.S. (November 1993) Lattice-based access control models. *IEEE Computer*, 26(11):9–19.
- Sandhu, R.S., E.J. Coyne, H.L. Feinstein and C.E. Youman (February 1996) Role-based access control models. *IEEE Computer*, 29(2):38–47.
- Schaefer, Marvin (2004) If A1 is the answer, what was the question? An edgy naïf's retrospective on promulgating the Trusted Computer Systems Evaluation Criteria. In *Proceedings of ACSAC 2004*, pages 204–228.
- Schellhorn, Gerhard, Wolfgang Reif, Axel Schairer, Paul Karger, Vernon Austel and David Toll (2000) Verification of a formal security model for multiapplicative smart cards. In F. Cuppens *et al.*, editors, *ESORICS 2000, LNCS 1895*, pages 17–36. Springer-Verlag.
- Schneider, Fred B. (2000) Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50.
- Schneier, B. (1996) *Applied Cryptography*. John Wiley & Sons, New York, 2nd edition.
- Schroeder, M. and J.H. Saltzer (1972) A hardware architecture for implementing protection rings. *Communications of the ACM*, 15(3):157–170.
- Shimomura, T. (1995) *Takedown*. Martin Secker & Warburg Ltd.
- Shoch, John F. and Jon A. Hupp (September 1982) The “worm” programs – early experience with a distributed computation. *Communications of the ACM*, 25(3):172–180.
- Shoup, Victor (2001) OAEP reconsidered. In J. Kilian, editor, *Advances in Cryptology – Crypto 2001, LNCS 2139*, pages 239–259. Springer-Verlag.
- Sibert, O., P.A. Porras and R. Lindell (1996) An analysis of the Intel 80x86 processor architecture and implementation. *IEEE Transactions on Software Engineering*, 22(5):283–293, May.
- Smith, Martin (1993) *Commonsense Computer Security*. McGraw-Hill, London.
- Spafford, E.H. (1989) Crisis and aftermath. *Communications of the ACM*, 32(6):678–687, June.
- Spitzner, Lance (2003) Honeypots. <http://www.tracking-hackers.com>, May.
- Stallings, William (2003) *Cryptography and Network Security*. Prentice Hall, 3rd edition.
- Sterne, Daniel F. (1991) On the buzzword “Security Policy”. In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*, pages 219–230.
- Stoll, C. (1989) *The Cuckoo's Egg*. Simon & Schuster.
- Swift, Michael M., Anne Hopkins, Peter Brundrett, Cliff Van Dyke, Praerit Garg, Shannon Chan, Mario Goertzel and Gregory Jensenworth (2002) Improving the granularity of access control for Windows 2000. *ACM Transactions on Information and System Security*, 5(4):398–437.
- UK ITSEC Scheme (1991) *Description of the Scheme*, March, UKSP 01.
- United Nations (1999) *International Review of Criminal Policy – United Nations Manual*

- on the Prevention and Control of Computer-related Crime*. New York.
- US Department of Commerce, National Bureau of Standards (1977) *Data Encryption Standard*, December, NBS FIPS PUB 46.
- US Department of Commerce, National Institute of Standards and Technology (2000) *Digital Signature Standard (DSS)*, January, FIPS PUB 186-2.
- US Department of Commerce, National Institute of Standards and Technology (2001) *Advanced Encryption Standard (AES)*, November, FIPS 197.
- US Department of Defense (1985) *DoD Trusted Computer System Evaluation Criteria*, DOD 5200.28-STD.
- US Department of Defense (1987) *Industrial Security Manual for Safeguarding Classified Information*, June, DOD 5220.22-M.
- Van der Putte, Ton and Jeroen Keuning (2000) Biometrical fingerprinting recognition: don't get your fingers burned. In Josep Domingo-Ferrer *et al.*, editors, *Smart Card Research and Applications*, pages 289-303. Kluwer Academic Publishers.
- Viega, John and Gary McGraw (2001) *Building Secure Software*. Addison-Wesley, Boston, MA.
- Viega, John and Matt Messier (2003) *Secure Programming Cookbook for C and C++*. O'Reilly & Associates.
- Wahbe, R., S. Lucco, T.E. Anderson and S.L. Graham (1993) Efficient software-based fault isolation. In *Proceedings of the Symposium on Operating System Principles*.
- Ware, W. (1979) Security controls for computer systems. Technical Report R-609-1, Rand Corp Tech Report, October.
- Weissman, C. (1992) BLACKER: security for the DDN, examples of A1 security engineering trades. In *Proceedings of the 1992 IEEE Symposium on Research in Security and Privacy*, pages 286-292.
- Wilkes, M.V. (1968) *Time-sharing Computer Systems*. Elsevier, New York.
- Wu, Thomas (1999) A real-world analysis of Kerberos password security. In *Proceedings of the 1999 Network and Distributed System Security Symposium*. Internet Society, February.
- Zwicky, Elizabeth D., Simon Cooper and D. Brent Chapman (1995) *Building Internet Firewalls*. O'Reilly & Associates, Sebastopol, CA, 2nd edition.